OpenModelica ユーザーズガイド和訳

リリース ***v1.14.1***

一般社団法人オープン **CAE** 学会

2020 年 **12** 月 **19** 日

目 次

Generated on 2020-12-19 at 23:11

このユーザガイド和訳は，一般社団法人 オープン CAE 学会の責任のもとで公開しております．本書に関するご意見等がございましたら，当学会事務局 (office@opencae.jp) までご連絡ください．本書の和訳は第1〜3 章となっています。

一般社団法人 オープン CAE 学会

This document is part of OpenModelica: https://www.openmodelica.org

Contact: OpenModelica@ida.liu.se

This users guide provides documentation and examples on how to use the OpenModelica system, both for the Modelica beginners and advanced users.

第1章　紹介

このドキュメントで説明する OpenModelica システムには、短期的な目標と長期的な目標があります。

- 短期的な目標は、Modelica 言語のための効率的でインタラクティブな計算環境を開発することと、Modelica 言語のほぼ完全な実装を行うことです。適切なツールとライブラリのサポートにより、Modelica は、低レベルと高レベル両方の数値アルゴリズムの開発と実行のための計算言語として非常に実用的であることがわかります。例えば、制御システムの設計、非線形方程式システムの解法、または複雑なアプリケーションに適用される最適化アルゴリズムの開発などが挙げられます。
- 長期的な目標は、Modelica 言語の完璧なリファレンスとなる実装です。方程式ベースのモデルのシミュレーションとプログラミング環境の追加機能、および言語設計内の研究、実験あるいはその他の研究活動のための便利な機能を実装することです。ただし、私たちの目標は、Modelica コンパイラによる高度な解析と最適化を必要とする大規模なモデルを取り扱うことができる現在の商用 Modelica ツールのような高いパフォーマンスとクオリティのレベルに到達することではありません。

Modelica 環境としてオープンソース OpenModelica の実装の長期的な **研究** 関連の目標と問題には、以下が含まれますが、これらに限定されません。

- 静的および動的セマンティクスの両方を含む Modelica の **完全に正式な仕様** の開発。そのような仕様は、リファレンス実装の一種として、セマンティックリファレンスを提供することによって現在および将来の Modelica の実装を支援するために使用することができます。
- **言語の設計** , 例えば言語の **スコープの拡張**。すなわち診断、構造解析、システム同定、偏微分方程式のような拡張が必要なモデリング問題、離散モデリングとシミュレーションのスコープの拡大など。
- 表現力、直交性、宣言に関する性質、再利用性、コンフィギュラ、アーキテクチャのような **抽象化に関するプロパティの改良** のための **言語の設計** 。
- **実装テクニックの改良** 、例えば、並列ハードウェアに向けたコード生成によって、コンパイルされた Modelica コードの性能を向上させること。
- モデル作成をより簡便にするため、Modelica のような方程式ベースの言語をサポートする **デバッキングの改良** 。
- 特定のアプリケーション領域のためのハイレベルな（グラフィカルな） **使いやすいユーザインタフェース** 。
- 結果の考察とプレゼンテーションのための **可視化** とアニメーション技術。
- 様々な応用分野の研究者による **アプリケーションの使用** と **モデルライブラリ** の開発。

OpenModelica 環境は言語設計のアイデアのためのテストベンチを提供します。もしそのテストベンチが成功した場合は、公式に Modelica の中に含める可能性を検討するため Modelica 協会に提出することも可能です。

OpenModelica 環境の現在のバージョンは、Modelica の式、アルゴリズム、および関数のほとんどが対話的に実行されることを可能にします。同様に方程式モデルと Modelica 関数は、効率的な C コードにコンパイルできます。生成された C コードは、ユーティリティ関数のライブラリ、ランタイム・ライブラリ、および DAE ソルバと組み合わされます。

1.1 システム概要

OpenModelica 環境は Figure 1.1 に示されるように、いくつかの相互接続するサブシステムから構成されています。

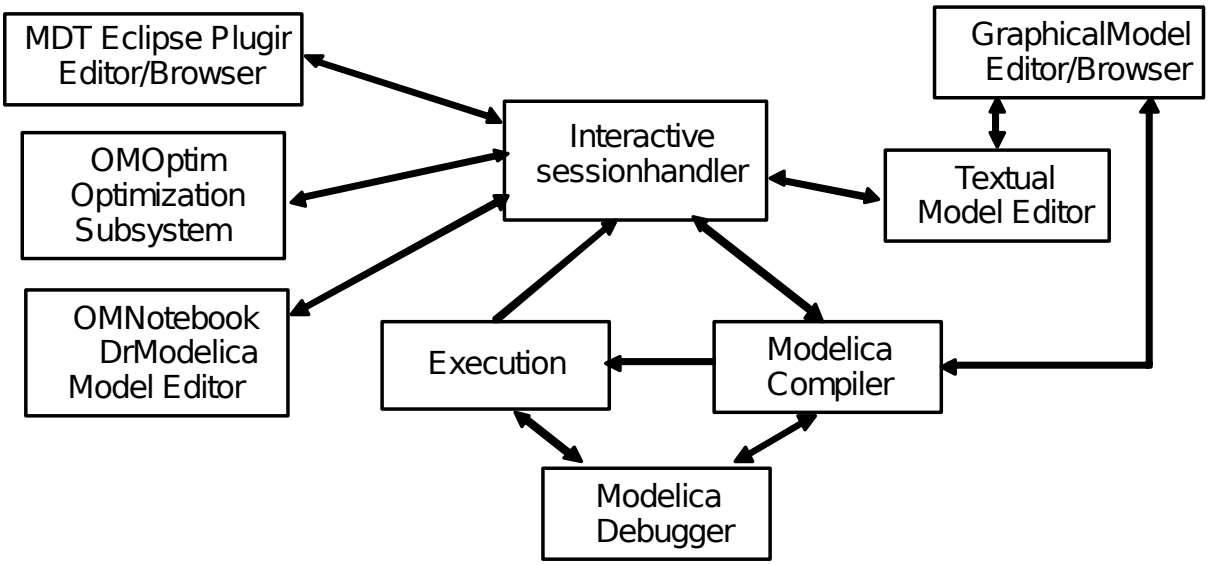


Figure 1.1: OpenModelica 環境のアーキテクチャ。矢印はデータ及び制御フローを示します。インタラクティブセッションハンドラはコマンドを受け取り、変換され実行されたコマンドや式から結果を表示します。 いくつかのサブシステムは、ブラウジングと Modelica コードのテキスト編集機能を提供します。デバッガは、現在 Modelica の拡張アルゴリズムのサブセットのデバッギングを提供します。

以下のサブシステムは現在 OpenModelica 環境に統合されています。

- *An interactive session handler* (インタラクティブセッションハンドラ) はコマンドと Modelica の計算式の評価、シミュレーション、プロットなどのパースとインタプリタです。セッションハンドラは単純なコマンドの履歴、およびファイル名の補完とコマンドの識別が含まれています。
- *A Modelica compiler subsystem* (Modelica コンパイラサブシステム) はクラス、関数、および変数の定義を含むシンボルテーブルを用いて Modelica コードを C コードに変換します。そのような定義は、事前にもしくはユーザーによって定義されているか、ライブラリから得ることができます。コンパイラは、インタプリタ型の使用方法と定数式の評価のための Modelica インタプリタを含んでいます。サブシステムは数値的な ODE ソルバーあるいは DAE ソルバーとリンクしたシミュレーション実行ファイルをビルドするための機能も含まれています。
- *An execution and run-time module* (実行とランタイムモジュール)。現在、このモジュールは、数値ソルバとリンクした方程式ベースのモデルに基づくシミュレーションコード及び、変換された式や関数からコンパイルされたバイナリコードを実行します。近い将来イベントを取り扱う機能は、Modelica 言語の離散化およびハイブリッド機能に含まれるでしょう。

- *Eclipse plugin editor/browser* (Eclipse のプラグインエディタ/ブラウザ)。 Eclipse のプラグインである MDT（Modelica 開発ツール）はファイルとクラスの階層構造の表示やテキスト編集機能を提供します。MDT は Emacs エディター/ブラウザーに類似しています。いくつかの構文ハイライト機能も含まれています。 Eclipse フレームワークは、簡単なリファクタリングや相互参照のサポートのような拡張を将来的に追加することも容易であるという利点を持っています。

- *OMNotebook DrModelica model editor* (OMNotebook DrModelica モデルエディタ)。このサブシステムは、高度な Mathematica ノートブックに比べて軽量で、MathModelica で利用可能なノートエディタを提供します。この基本的な機能は DrModelica チュートリアルで扱うことができます。基本的なフォーマットを含み章と節を持つ階層的な文章を表示し編集することができます。セルは一般的なテキストや Modelica モデルと式を含めることができます。その Modelica モデルはコンパイルし計算することが出来ます。しかし、数学的な組版機能は、このノートブックエディタのセルではまだ利用できません。

- *Graphical model editor/browser OMEdit* (グラフィカルなモデルエディタ/ブラウザ OMEdit)。これは Modelica クラスのインスタンスを接続し Modelica モデルライブラリをブラウズしてピッキングすることができるコンポーネントベースのモデル設計のためのグラフィカルなエディタです。グラフィカルモデルエディタは、モデルクラスを編集するためのテキストエディタ、およびインタラクティブな Modelica コマンドの評価を表示するためのウィンドウを含んでいます。

- *Optimization subsystem OMOptim* (最適化サブシステム OMOptim)。これは OpenModelica のための最適化サブシステムで、モデルの設計パラメータの最適な組合せを選択するためのものです。現在のバージョンはグラフィカル・ユーザー・インターフェースがあり、遺伝的最適化アルゴリズムとパレートフロント最適化を有し、シミュレータと統合され、自動的に Modelica モデルから変数および設計パラメータにアクセスします。

- *Dynamic Optimization subsystem* (動的最適化サブシステム)。これはコロケーション法を使用した動的最適化です。この機能は目標関数と制約条件を有する、最適化の仕様に基づいて継承された Modelica モデルのためのものです。このサブシステムは、OpenModelica コンパイラに統合されています。

- *Modelica equation model debugger* (Modelica equation モデルデバッガー)。equation セクションに対するデバッガーは、ソースコード内のエラーの位置を示します。これは、コンパイラによって方程式から低レベルな C コードに変換された際のシンボリックな変換を追跡しています。また、実施された変換内容について説明しています。

- *Modelica algorithmic code debugger* (Modelica algorithm コードデバッガ)。Modelica algorithm コードデバッガは、拡張されたアルゴリズムサブセットに対するデバッギング機能を提供します。この機能のデバッグの対象には方程式ベースのモデルおよびいくつかの機能は含まれていません。しかし、いくつかのメタプログラミングと Modelica へのモデル変換を含みます。これは、Eclipse を使用しておりソースコードのステップ実行機能、ブレークポイントの設定などの一般的なデバッギング機能を全て有するデバッガです。またこのデバッガには、継承された Modelica でツリー構造またはリスト構造のような階層データを閲覧するためのデータ・ビューブラウザが含まれています。

1.2 インタラクティブセッションを用いたサンプル

以下は OMShell(OpenModelica シェル) と呼ばれる OpenModelica 環境における対話的なセッションハンドラを使用するインタラクティブなセッションです。以降の例のほとんどはテストモデルと同様に *OMNotebook with DrModelica and DrControl* UsersGuideExamples.onb の中でご利用いただけます。

```
>>> getInstallationDirectoryPath() + "/share/doc/omc/testmodels/"
"<<OPENMODELICAHOME>>/share/doc/omc/testmodels/"
```

以下のコマンドは以下のバージョンの OpenModelica を用いて実行されています

```
>>> getVersion()
"OMCompiler v1.14.1"
```

1.2.1 インタラクティブセッションの開始

Windows 版の場合、インタラクティブウィンドウはスタートメニューから OpenModelica -> OpenModelica Shell を選択することで起動します。

変数 x に範囲 1:12 のベクトルを入力します。計算式を実行すると値が返されます。

```
>>> x := 1:12
{1,2,3,4,5,6,7,8,9,10,11,12}
```

1.2.2 インタラクティブモードの使用

OMShell を使ってインスタンスを作るためインタラクティブモードで OMC(OpenModelica Compiler) を実行する場合、クラスをロードしコマンドを実行します。ここではいくつかの例となるセッションを示します。

例. セッション 1

```
>>> model A Integer t = 1.5; end A; //The type is Integer but 1.5 is of Real Type
{A}
>>> instantiateModel(A)
""
"[<interactive>:1:9-1:23:writable] Error: Type mismatch in binding t = 1.5,␣
↪expected subtype of Integer, got type Real.
Error: Error occurred while flattening model A
"
```

例. セッション 2

サンプルを実行してエラーメッセージが表示されない場合、getErrorString() コマンドを実行してください。

```
model C
  Integer a;
  Real b;
equation
  der(a) = b;
  der(b) = 12.0;
end C;
```

```
>>> instantiateModel(C)
""
```

エラー:

[<interactive>:5:3-5:13:writable] Error: Argument 'a' to der has illegal type Integer, must be a subtype of Real.

Error: Error occurred while flattening model C

1.2.3 バブルソートファンクションを試す

プルダウンメニューの File - >Load Model を選択するか、明示的にコマンドを入力することによって、バブルソートファンクションをロードします。

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/
↪bubblesort.mo")
true
```

バブルソートファンクションは降順にベクトル x を並べるために次のようにコールされます。ソートされた結果は、それらの型と一緒に返されます。結果のベクトルは Real[:] 型であることに注意してください。x は function の戻り値として宣言された型のため Real[12] となります。Interger 型ベクトルが入力されても Modelica の型の強制的なルールに従って自動的に Real 型ベクトルに変換されます。この function が実行される前に呼び出されたときに自動的にコンパイルされます。

```
>>> bubblesort(x)
{12.0,11.0,10.0,9.0,8.0,7.0,6.0,5.0,4.0,3.0,2.0,1.0}
```

その他のコール

```
>>> bubblesort({4,6,2,5,8})
{8.0,6.0,5.0,4.0,2.0}
```

1.2.4 system コマンドと cd コマンドを試す

システムユーティリティ機能を経由してオペレーティングシステムのコマンドを実行することも可能です。コマンドは、文字列引数として与えます。以下の例では、システムユーティリティは UNIX コマンドの cat を実行しています。ここで cat コマンドは OMC からコマンドラインを通して bubblesort.mo ファイルの内容を出力します。

```
>>> system("cat '"+getInstallationDirectoryPath()+"/share/doc/omc/testmodels/
↪bubblesort.mo' > bubblesort.mo")
0
```

```
function bubblesort
  input Real[:] x;
  output Real[size(x,1)] y;
protected
  Real t;
algorithm
  y := x;
  for i in 1:size(x,1) loop
    for j in 1:size(x,1) loop
      if y[i] > y[j] then
        t := y[i];
        y[i] := y[j];
        y[j] := t;
      end if;
    end for;
  end for;
end bubblesort;
```

注：system コマンドによって標準出力に送られた出力は、CORBA ベースのクライアントが実行されているとき GUI のウィンドウではなくログ・ファイルに入れられます。したがって、上記の cat コマンドによって出力されたテキストは GUI ウィンドウには表示されません。そのため上記の例では別のファイルにリダイレクトしています。

ファイル読込みは readFile コマンドを使用します。

```
>>> readFile("bubblesort.mo")
function bubblesort
  input Real[:] x;
  output Real[size(x,1)] y;
protected
  Real t;
algorithm
  y := x;
  for i in 1:size(x,1) loop
    for j in 1:size(x,1) loop
      if y[i] > y[j] then
        t := y[i];
        y[i] := y[j];
        y[j] := t;
      end if;
    end for;
  end for;
end bubblesort;
```

system コマンドはサクセスコードのみを返します（成功した場合は 0 を返します)

```
>>> system("dir")
0
>>> system("Non-existing command")
127
```

またその他のビルトインコマンドとして cd コマンド（**カレントディレクトリの変更**）があります。実行結果として得られたカレントディレクトリの情報は文字列として返されます。

```
>>> dir:=cd()
"<<DOCHOME>>"
>>> cd("source")
"<<DOCHOME>>/source"
>>> cd(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/")
"/home/OpenModelica/build/share/doc/omc/testmodels"
>>> cd(dir)
"<<DOCHOME>>"
```

1.2.5 Modelica ライブラリと DC モータモデル

メニューアイテムの File - > Load Modelica Library から Modelica Standard Library 全体を読み込むことが出来ます。

```
>>> loadModel(Modelica)
true
```

dcmotor モデルが入ったファイルをロードします。

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/dcmotor.mo
↪")
true
```

警告:

Warning: Requested package Modelica of version 3.2.2, but this package was already loaded with version 3.2.3. You might experience problems if these versions are incompatible.

モデルをシミュレートします

```
>>> simulate(dcmotor, startTime=0.0, stopTime=10.0)
record SimulationResult
    resultFile = "<<DOCHOME>>/dcmotor_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 10.0, numberOfIntervals = 500,
 tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'dcmotor', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.6879983999999999,
    timeBackend = 0.03004,
    timeSimCode = 0.007973900000000001,
    timeTemplates = 0.018141,
```

(次のページに続く)

(前のページからの続き)

```
    timeCompile = 2.5703431,
    timeSimulation = 0.1166419,
    timeTotal = 3.4335717
end SimulationResult;
```

警告:

Warning: Requested package Modelica of version 3.2.2, but this package was already loaded with version 3.2.3. You might experience problems if these versions are incompatible.

モデルのソースコードをリストします

```
>>> list(dcmotor)
model dcmotor
  import Modelica.Electrical.Analog.Basic;
  Basic.Resistor resistor1(R = 10);
  Basic.Inductor inductor1(L = 0.2, i.fixed = true);
  Basic.Ground ground1;
  Modelica.Mechanics.Rotational.Components.Inertia load(J = 1, phi.fixed = true, w.
↪fixed = true);
  Basic.EMF emf1(k = 1.0);
  Modelica.Blocks.Sources.Step step1;
  Modelica.Electrical.Analog.Sources.SignalVoltage signalVoltage1;
equation
  connect(step1.y, signalVoltage1.v);
  connect(signalVoltage1.p, resistor1.p);
  connect(resistor1.n, inductor1.p);
  connect(inductor1.n, emf1.p);
  connect(emf1.flange, load.flange_a);
  connect(signalVoltage1.n, ground1.p);
  connect(ground1.p, emf1.n);
  annotation(
    uses(Modelica(version = "3.2.2")));
end dcmotor;
```

フラットなコードへ変更するためモデルのインスタンス化を試します。

```
>>> instantiateModel(dcmotor)
class dcmotor
  Real resistor1.v(quantity = "ElectricPotential", unit = "V") "Voltage drop of␣
↪the two pins (= p.v - n.v)";
  Real resistor1.i(quantity = "ElectricCurrent", unit = "A") "Current flowing from␣
↪pin p to pin n";
  Real resistor1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at the␣
↪pin";
  Real resistor1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing␣
↪into the pin";
  Real resistor1.n.v(quantity = "ElectricPotential", unit = "V") "Potential at the␣
↪pin";
  Real resistor1.n.i(quantity = "ElectricCurrent", unit = "A") "Current flowing␣
↪into the pin";
  parameter Boolean resistor1.useHeatPort = false "=true, if heatPort is enabled";
  parameter Real resistor1.T(quantity = "ThermodynamicTemperature", unit = "K",␣
↪displayUnit = "degC", min = 0.0, start = 288.15, nominal = 300.0) = resistor1.T_
↪ref "Fixed device temperature if useHeatPort = false";
  Real resistor1.LossPower(quantity = "Power", unit = "W") "Loss power leaving␣
↪component via heatPort";
```

(次のページに続く)

(前のページからの続き)

```
 Real resistor1.T_heatPort(quantity = "ThermodynamicTemperature", unit = "K",
displayUnit = "degC", min = 0.0, start = 288.15, nominal = 300.0) "Temperature of␣
↪heatPort";
 parameter Real resistor1.R(quantity = "Resistance", unit = "Ohm", start = 1.0) =␣
↪10.0 "Resistance at temperature T_ref";
 parameter Real resistor1.T_ref(quantity = "ThermodynamicTemperature", unit = "K",
displayUnit = "degC", min = 0.0, start = 288.15, nominal = 300.0) = 300.15
↪"Reference temperature";
 parameter Real resistor1.alpha(quantity = "LinearTemperatureCoefficient", unit =
↪"1/K") = 0.0 "Temperature coefficient of resistance (R_actual = R*(1 + alpha*(T_
↪heatPort - T_ref))";
 Real resistor1.R_actual(quantity = "Resistance", unit = "Ohm") "Actual␣
↪resistance = R*(1 + alpha*(T_heatPort - T_ref))";
 Real inductor1.v(quantity = "ElectricPotential", unit = "V") "Voltage drop of␣
↪the two pins (= p.v - n.v)";
 Real inductor1.i(quantity = "ElectricCurrent", unit = "A", start = 0.0, fixed =␣
↪true) "Current flowing from pin p to pin n";
 Real inductor1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at the␣
↪pin";
 Real inductor1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing␣
↪into the pin";
 Real inductor1.n.v(quantity = "ElectricPotential", unit = "V") "Potential at the␣
↪pin";
 Real inductor1.n.i(quantity = "ElectricCurrent", unit = "A") "Current flowing␣
↪into the pin";
 parameter Real inductor1.L(quantity = "Inductance", unit = "H", start = 1.0) = 0.
↪2 "Inductance";
 Real ground1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at the␣
↪pin";
 Real ground1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into␣
↪the pin";
 Real load.flange_a.phi(quantity = "Angle", unit = "rad", displayUnit = "deg")
↪"Absolute rotation angle of flange";
 Real load.flange_a.tau(quantity = "Torque", unit = "N.m") "Cut torque in the␣
↪flange";
 Real load.flange_b.phi(quantity = "Angle", unit = "rad", displayUnit = "deg")
↪"Absolute rotation angle of flange";
 Real load.flange_b.tau(quantity = "Torque", unit = "N.m") "Cut torque in the␣
↪flange";
 parameter Real load.J(quantity = "MomentOfInertia", unit = "kg.m2", min = 0.0,␣
↪start = 1.0) = 1.0 "Moment of inertia";
 parameter enumeration(never, avoid, default, prefer, always) load.stateSelect =␣
↪StateSelect.default "Priority to use phi and w as states";
 Real load.phi(quantity = "Angle", unit = "rad", displayUnit = "deg", fixed =␣
↪true, stateSelect = StateSelect.default) "Absolute rotation angle of component";
 Real load.w(quantity = "AngularVelocity", unit = "rad/s", fixed = true,␣
↪stateSelect = StateSelect.default) "Absolute angular velocity of component (=␣
↪der(phi))";
 Real load.a(quantity = "AngularAcceleration", unit = "rad/s2") "Absolute angular␣
↪acceleration of component (= der(w))";
 parameter Boolean emf1.useSupport = false "= true, if support flange enabled,␣
↪otherwise implicitly grounded";
 parameter Real emf1.k(quantity = "ElectricalTorqueConstant", unit = "N.m/A",␣
↪start = 1.0) = 1.0 "Transformation coefficient";
 Real emf1.v(quantity = "ElectricPotential", unit = "V") "Voltage drop between␣
↪the two pins";
 Real emf1.i(quantity = "ElectricCurrent", unit = "A") "Current flowing from␣
↪positive to negative pin";
 Real emf1.phi(quantity = "Angle", unit = "rad", displayUnit = "deg") "Angle of␣
↪shaft flange with respect to support (= flange.phi - support.phi)";
 Real emf1.w(quantity = "AngularVelocity", unit = "rad/s") "Angular velocity of␣
↪flange relative to support";
```

(次のページに続く)

(前のページからの続き)

```
  Real emf1.tau(quantity = "Torque", unit = "N.m") "Torque of flange";
  Real emf1.tauElectrical(quantity = "Torque", unit = "N.m") "Electrical torque";
  Real emf1.p.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real emf1.p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into␣
↪the pin";
  Real emf1.n.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real emf1.n.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into␣
↪the pin";
  Real emf1.flange.phi(quantity = "Angle", unit = "rad", displayUnit = "deg")
↪"Absolute rotation angle of flange";
  Real emf1.flange.tau(quantity = "Torque", unit = "N.m") "Cut torque in the flange
↪";
  protected Real emf1.internalSupport.tau(quantity = "Torque", unit = "N.m") = -
↪emf1.tau "External support torque (must be computed via torque balance in model␣
↪where InternalSupport is used; = flange.tau)";
  protected Real emf1.internalSupport.phi(quantity = "Angle", unit = "rad",␣
↪displayUnit = "deg") "External support angle (= flange.phi)";
  protected Real emf1.internalSupport.flange.phi(quantity = "Angle", unit = "rad",␣
↪displayUnit = "deg") "Absolute rotation angle of flange";
  protected Real emf1.internalSupport.flange.tau(quantity = "Torque", unit = "N.m
↪") "Cut torque in the flange";
  protected parameter Real emf1.fixed.phi0(quantity = "Angle", unit = "rad",␣
↪displayUnit = "deg") = 0.0 "Fixed offset angle of housing";
  protected Real emf1.fixed.flange.phi(quantity = "Angle", unit = "rad",␣
↪displayUnit = "deg") "Absolute rotation angle of flange";
  protected Real emf1.fixed.flange.tau(quantity = "Torque", unit = "N.m") "Cut␣
↪torque in the flange";
  Real step1.y "Connector of Real output signal";
  parameter Real step1.offset = 0.0 "Offset of output signal y";
  parameter Real step1.startTime(quantity = "Time", unit = "s") = 0.0 "Output y =␣
↪offset for time < startTime";
  parameter Real step1.height = 1.0 "Height of step";
  Real signalVoltage1.p.v(quantity = "ElectricPotential", unit = "V") "Potential␣
↪at the pin";
  Real signalVoltage1.p.i(quantity = "ElectricCurrent", unit = "A") "Current␣
↪flowing into the pin";
  Real signalVoltage1.n.v(quantity = "ElectricPotential", unit = "V") "Potential␣
↪at the pin";
  Real signalVoltage1.n.i(quantity = "ElectricCurrent", unit = "A") "Current␣
↪flowing into the pin";
  Real signalVoltage1.v(unit = "V") "Voltage between pin p and n (= p.v - n.v) as␣
↪input signal";
  Real signalVoltage1.i(quantity = "ElectricCurrent", unit = "A") "Current flowing␣
↪from pin p to pin n";
equation
  assert(1.0 + resistor1.alpha * (resistor1.T_heatPort - resistor1.T_ref) >= 1e-15,
 "Temperature outside scope of model!");
  resistor1.R_actual = resistor1.R * (1.0 + resistor1.alpha * (resistor1.T_
↪heatPort - resistor1.T_ref));
  resistor1.v = resistor1.R_actual * resistor1.i;
  resistor1.LossPower = resistor1.v * resistor1.i;
  resistor1.v = resistor1.p.v - resistor1.n.v;
  0.0 = resistor1.p.i + resistor1.n.i;
  resistor1.i = resistor1.p.i;
  resistor1.T_heatPort = resistor1.T;
  inductor1.L * der(inductor1.i) = inductor1.v;
  inductor1.v = inductor1.p.v - inductor1.n.v;
  0.0 = inductor1.p.i + inductor1.n.i;
  inductor1.i = inductor1.p.i;
  ground1.p.v = 0.0;
  load.phi = load.flange_a.phi;
```

(次のページに続く)

(前のページからの続き)

```
  load.phi = load.flange_b.phi;
  load.w = der(load.phi);
  load.a = der(load.w);
  load.J * load.a = load.flange_a.tau + load.flange_b.tau;
  emf1.internalSupport.flange.tau = emf1.internalSupport.tau;
  emf1.internalSupport.flange.phi = emf1.internalSupport.phi;
  emf1.fixed.flange.phi = emf1.fixed.phi0;
  emf1.v = emf1.p.v - emf1.n.v;
  0.0 = emf1.p.i + emf1.n.i;
  emf1.i = emf1.p.i;
  emf1.phi = emf1.flange.phi - emf1.internalSupport.phi;
  emf1.w = der(emf1.phi);
  emf1.k * emf1.w = emf1.v;
  emf1.tau = (-emf1.k) * emf1.i;
  emf1.tauElectrical = -emf1.tau;
  emf1.tau = emf1.flange.tau;
  step1.y = step1.offset + (if time < step1.startTime then 0.0 else step1.height);
  signalVoltage1.v = signalVoltage1.p.v - signalVoltage1.n.v;
  0.0 = signalVoltage1.p.i + signalVoltage1.n.i;
  signalVoltage1.i = signalVoltage1.p.i;
  resistor1.p.i + signalVoltage1.p.i = 0.0;
  resistor1.n.i + inductor1.p.i = 0.0;
  inductor1.n.i + emf1.p.i = 0.0;
  ground1.p.i + emf1.n.i + signalVoltage1.n.i = 0.0;
  load.flange_a.tau + emf1.flange.tau = 0.0;
  load.flange_b.tau = 0.0;
  emf1.fixed.flange.tau + emf1.internalSupport.flange.tau = 0.0;
  emf1.fixed.flange.phi = emf1.internalSupport.flange.phi;
  signalVoltage1.v = step1.y;
  resistor1.p.v = signalVoltage1.p.v;
  inductor1.p.v = resistor1.n.v;
  emf1.p.v = inductor1.n.v;
  emf1.flange.phi = load.flange_a.phi;
  emf1.n.v = ground1.p.v;
  emf1.n.v = signalVoltage1.n.v;
end dcmotor;
```

警告:

Warning: Requested package Modelica of version 3.2.2, but this package was already loaded with version 3.2.3. You might experience problems if these versions are incompatible.

シミュレーション結果の一部をプロットします

1.2.6 val() ファンクション

val(*variableName*,*time*) スクリプトファンクションは、シミュレーションのある時点でのシミュレーション結果の変数の補間値を取得するために使用することができます。使用方法は以下のバウンシングボールのシミュレーションをご参照ください。

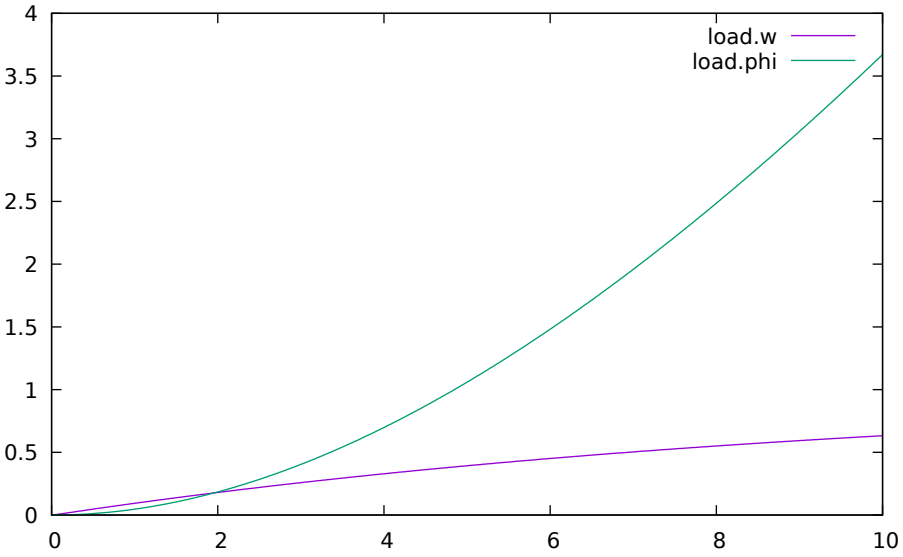


Figure 1.2: DC モータの回転角と回転速度

1.2.7 バウンシングボールとスイッチモデル

when-方程式と if-文（Modelica キーワードは見やすくするため太字としています）を含むバウンシングボールのサンプルをロードしシミュレートします:

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/
↪BouncingBall.mo")
true
```

```
>>> list(BouncingBall)
model BouncingBall
  parameter Real e = 0.7 "coefficient of restitution";
  parameter Real g = 9.81 "gravity acceleration";
  Real h(fixed = true, start = 1) "height of ball";
  Real v(fixed = true) "velocity of ball";
  Boolean flying(fixed = true, start = true) "true, if ball is flying";
  Boolean impact;
  Real v_new(fixed = true);
  Integer foo;
equation
  impact = h <= 0.0;
  foo = if impact then 1 else 2;
  der(v) = if flying then -g else 0;
  der(h) = v;
  when {h <= 0.0 and v <= 0.0, impact} then
    v_new = if edge(impact) then -e * pre(v) else 0;
    flying = v_new > 0;
    reinit(v, v_new);
  end when;
end BouncingBall;
```

simulate と plot コマンドを実行する代わりに、これらのコマンドが含まれている sim_BouncingBall.mos（Modelica スクリプト）ファイルを runScript コマンドで実行します。

```
>>> writeFile("sim_BouncingBall.mos", "
  loadFile(getInstallationDirectoryPath() + \"/share/doc/omc/testmodels/
↪BouncingBall.mo\");
  simulate(BouncingBall, stopTime=3.0);
  /* plot({h,flying}); */
")
true
>>> runScript("sim_BouncingBall.mos")
"true
record SimulationResult
    resultFile = \"<<DOCHOME>>/BouncingBall_res.mat\",
    simulationOptions = \"startTime = 0.0, stopTime = 3.0, numberOfIntervals = 500,
 tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'BouncingBall', options = '
↪', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''\",
    messages = \"LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
\",
    timeFrontend = 0.0067438,
    timeBackend = 0.0067656,
    timeSimCode = 0.0016182,
    timeTemplates = 0.0122438,
    timeCompile = 2.508196,
    timeSimulation = 0.1280165999999999,
    timeTotal = 2.6654324
end SimulationResult;
"
```

```
model Switch
  Real v;
  Real i;
  Real i1;
  Real itot;
  Boolean open;
equation
  itot = i + i1;
  if open then
    v = 0;
  else
    i = 0;
  end if;
  1 - i1 = 0;
  1 - v - i = 0;
  open = time >= 0.5;
end Switch;
```

```
>>> simulate(Switch, startTime=0, stopTime=1)
record SimulationResult
    resultFile = "<<DOCHOME>>/Switch_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 500,␣
↪tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'Switch', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.00589,
    timeBackend = 0.0180014,
    timeSimCode = 0.0038736,
    timeTemplates = 0.0121149,
```

(次のページに続く)

(前のページからの続き)

```
    timeCompile = 2.5831589,
    timeSimulation = 0.1173981,
    timeTotal = 2.7422009
end SimulationResult;
```

val(variableName, time) ファンクションを用いて、time= 0 で itot の値を取得します。

```
>>> val(itot,0)
1.0
```

itot と open のプロット:

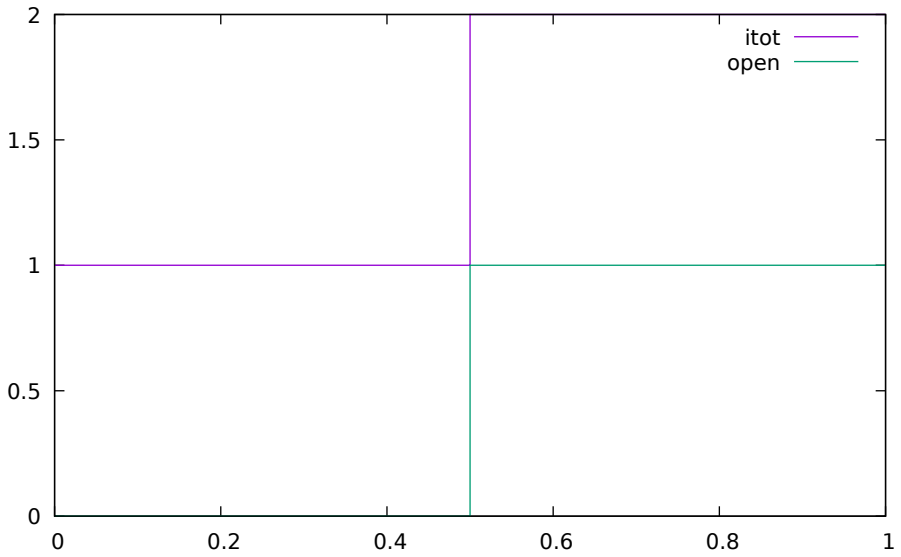


Figure 1.3: スイッチが開いた時のプロット

変数 open が false（0）から true（1）に変化し、itot が 1.0 から 2.0 に増加したことに注意してください。

1.2.8 すべてのモデルのクリア

全ての読込済みライブラリとモデルのクリアを行います

```
>>> clear()
true
```

読込済みモデルをリストして何も残っていないことを確認してください。

```
>>> list()
""
```

1.2.9 ファン・デル・ポールモデルとパラメトリックプロット

VanDerPol モデル（またはメニューから File - > Load Model）を読み込みます

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/VanDerPol.
↪mo")
true
```

モデルをシミュレートします

```
>>> simulate(VanDerPol, stopTime=80)
record SimulationResult
    resultFile = "<<DOCHOME>>/VanDerPol_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 80.0, numberOfIntervals = 500,
 tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'VanDerPol', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.0410226,
    timeBackend = 0.0029176,
    timeSimCode = 0.0008391000000000001,
    timeTemplates = 0.010602,
    timeCompile = 2.5284916,
    timeSimulation = 0.1132794,
    timeTotal = 2.699363
end SimulationResult;
```

プロットします。

```
>>> plotParametric("x","y")
```

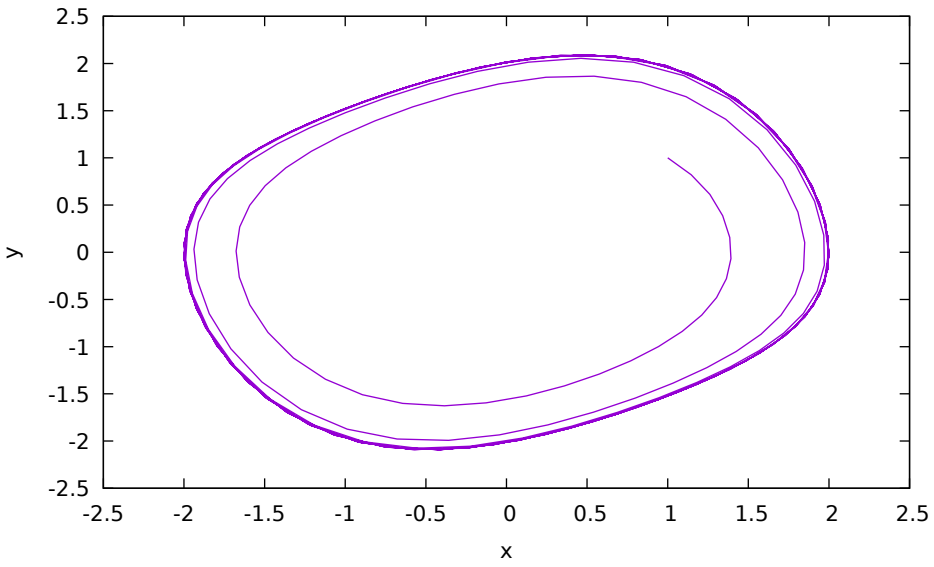


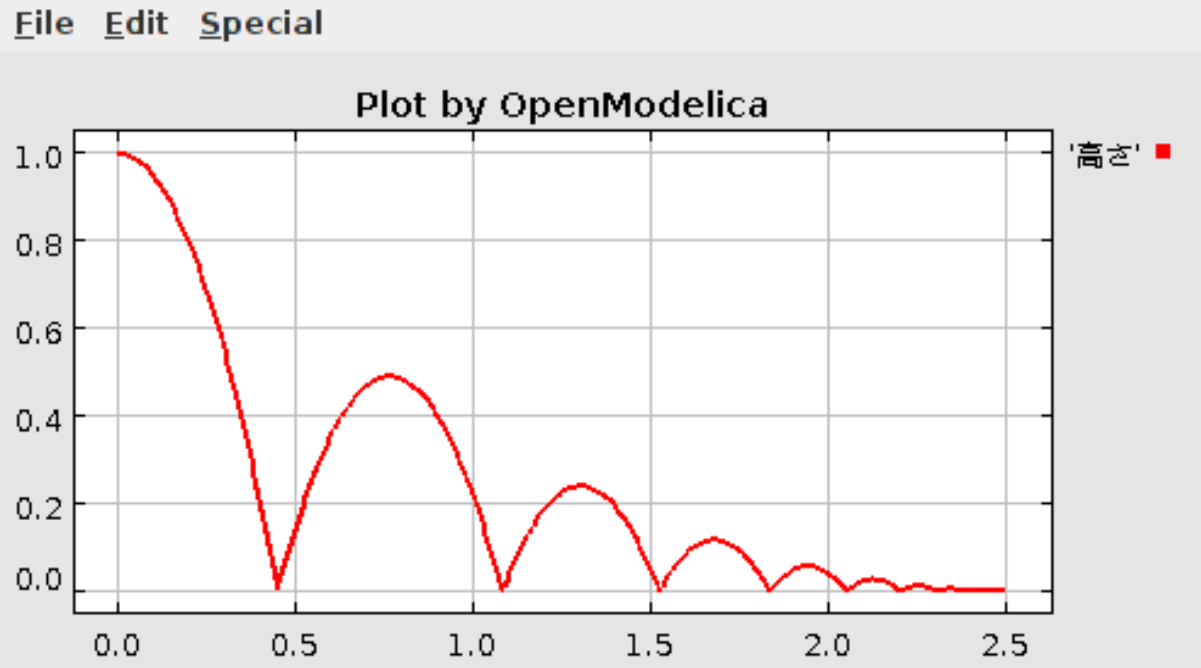
Figure 1.4: VanDerPol のパラメトリックプロット (x, y)

VanDerPol モデルをフラットなコードにするためインスタンス化の実行

```
>>> instantiateModel(VanDerPol)
class VanDerPol "Van der Pol oscillator model"
  Real x(start = 1.0, fixed = true);
  Real y(start = 1.0, fixed = true);
  parameter Real lambda = 0.3;
equation
  der(x) = y;
  der(y) = lambda * (1.0 - x ^ 2.0) * y - x;
end VanDerPol;
```

1.2.10 日本語あるいは中国語の文字の使用

日本語、中国語、および他の種類の Unicode 文字は、シングルクォートで囲まれた識別子内で使用することができます。以下のプロットウィンドウの右側の変数名の例を参照してください。



1.2.11 For ループ、While ループ、if ステートメントを用いたスクリプト

簡単な整数の合計のループ (OMShell の中で各行を個別に評価しないで複数行をインプットするには他のドキュメントからコピー&ペーストしてください)

```
>>> k := 0;
>>> for i in 1:1000 loop
  k := k + i;
end for;
>>> k
500500
```

実数と整数を使ったネストしたループ

```
>>> g := 0.0;
>>> h := 5;
>>> for i in {23.0,77.12,88.23} loop
  for j in i:0.5:(i+1) loop
    g := g + j;
    g := g + h / 2;
  end for;
  h := h + g;
end for;
```

セミコロンで区切られた 2 つ（またはそれ以上）の変数や代入文を置くことによって、2 つ以上の変数の値を確認することができます。この操作は変数で終わる必要があります。

```
>>> h; g
1997.45
1479.09
```

ベクトルを横断して文字要素を連結する for ループ

```
>>> i:="";
>>> lst := {"Here ", "are ","some ","strings."};
>>> s := "";
>>> for i in lst loop
  s := s + i;
end for;
>>> s
"Here are some strings."
```

一般的な while ループを用いた"abs"文字列 10 個の連結

```
>>> s:="";
>>> i:=1;
>>> while i<=10 loop
  s:="abc "+s;
  i:=i+1;
end while;
>>> s
"abc abc abc abc abc abc abc abc abc abc "
```

簡単な if ステートメントの例です。セミコロンの後ろの文末に変数を置くことによって、その値は計算された後に返されます。

```
>>> if 5>2 then a := 77; end if; a
77
```

elseif を持つ if-then-else ステートメント

```
>>> if false then
  a := 5;
elseif a > 50 then
  b:= "test"; a:= 100;
else
  a:=34;
end if;
```

変数 a と b を確認してください。

```
>>> a;b
100
"test"
```

1.2.12 変数、ファンクションと変数の型

変数にベクトルを割り当てます。

```
>>> a:=1:5
{1,2,3,4,5}
```

ファンクションで入力します。

```
function mySqr
  input Real x;
  output Real y;
algorithm
  y:=x*x;
end mySqr;
```

ファンクションを呼び出します。

```
>>> b:=mySqr(2)
4.0
```

変数 a の値を確認してください：

```
>>> a
{1,2,3,4,5}
```

a の型を確認してください。

```
>>> typeOf(a)
"Integer[5]"
```

b の型を取得します。

```
>>> typeOf(b)
"Real"
```

mySqr の型は何でしょうか？残念ながら現在、関数の型の確認は出来ません。

```
>>> typeOf(mySqr)
```

利用可能な変数をリストします。

```
>>> listVariables()
{b,a,s,lst,i,h,g,k,currentSimulationResult}
```

再びクリアします：

```
>>> clear()
true
```

1.2.13 エラーの原因に関する情報の取得

シミュレーションの失敗後、エラーの原因についてさらに詳しい情報を得るために getErrorString() ファンクションをコールします。

```
>>> getErrorString()
""
```

1.2.14 シミュレーション出力形式の変更

シミュレーションの出力形式にはいくつかの出力形式があり、mat 形式がデフォルトです。シミュレーション後に val() と plot() ファンクションが使用できるのは plt と mat だけです。plt と mat の速度を比較すると mat は軽いファイルに対して 5 倍程度速いです。そして mat はバイナリ形式のため大きなファイルに対してはよりスケールします。データ量の多いシミュレーションの場合、csv 形式は plt の約 2 倍の速度です。plt 形式は、シミュレーション中にすべての出力データを RAM に割り当てます。つまり、アプリケーションが 32 ビットプラットフォームで 4GB のメモリしかアドレス指定できないため、シミュレーションが失敗する可能性があります。Empty は出力をまったく行わないため、はるかに高速です。csv および plt 形式は、外部スクリプトまたは gnuplot などのツールを使用してプロットを生成したりデータを処理したりする場合に適しています。 mat 形式は、 MATLAB あるいは Octave で後処理できます。

```
>>> simulate(... , outputFormat="mat")
>>> simulate(... , outputFormat="csv")
>>> simulate(... , outputFormat="plt")
>>> simulate(... , outputFormat="empty")
```

結果ファイルにどの変数を含めるかを指定することもできます。これは、POSIX Extended Regular Expressions を使用して行われます。指定された式は、完全に変数名と一致しなければなりません（^および$記号は、指定された正規表現に自動的に追加されます）。

// デフォルトです。すべての変数がマッチします。

```
>>> simulate(... , variableFilter=".*")
```

//文字 1 から 3 の組み合わせを使用して、数字のみを含む変数 myVar のインデックスと一致します。

// of the letters 1 through 3

```
>>> simulate(... , variableFilter="myVar\\[[1-3]*\\]")
```

x, y または z と一致します。

```
>>> simulate(... , variableFilter="x|y|z")
```

1.2.15 外部ファンクションの使用

他のプログラミング言語で書かれた関数を呼び出す方法は *Interoperability – C and Python* を参照してください。

1.2.16 OpenMP のマルチコアサポートを用いた並列シミュレーションの使用

マルチコアコンピューター上のより高速なシミュレーションは、連立方程式を自動的に分割し、共有メモリ OpenMP を使用してさまざまなコアで実行するようにパーツを管理する新しい OpenModelica の機能を使用して得られます。並列計算の効果は、連立方程式をうまく分割できるかどうかというモデル構造に依存します。現在の OpenModelica のバージョンは負荷分散のない実験的なバージョンです。次のコマンドは、OpenModelica GUI からはまだ使用できませんが並列計算を実行します。

```
>>> omc -d=openmp model.mo
```

1.2.17 特定のバージョンのライブラリの読み込み

Modelic Standard Library には様々なバージョンがあり、それらはお互いに互換性はありません。getModelicaPath（）を呼び出すことで指定されたディレクトリに、同じライブラリの複数のバージョンを保存しておくことができます。loadModel(Modelica,{3.2"}) を呼び出すことにより、OpenModelica は"Modelica 3.2"というディレクトリまたは"Modelica 3.2.mo"というファイルを検索します。複数のライブラリバージョンを検索することができます。 プレリリースバージョンのライブラリがインストールされている場合は、そのライブラリを優先して検索します。 検索されたバージョンが"default"の場合、優先順位は次のとおりです。バージョン名なし"（Modelica）、メインリリースバージョン（Modelica 3.1）、プレリリースバージョン（Modelica 3.1Beta 1）、およびオーダーされていないバージョン（Modelica Special Release）。

loadModel コマンドは、ロードした後にトップ階層のクラスの uses アノテーションも調べます。 次のパッケージを指定すると、Complex1.0 および ModelicaServices1.1 も AST に自動的にロードされます。

```
package Modelica
  annotation(uses(Complex(version="1.0"),
  ModelicaServices(version="1.1")));
end Modelica;
```

```
>>> clear()
true
```

モデルに uses アノテーションがある場合も、パッケージがロードされます。

```
model M
  annotation(uses(Modelica(version="3.2.1")));
end M;
```

```
>>> instantiateModel(M)
class M
end M;
```

注釈:

Notification: Automatically loaded package Modelica 3.2.1 due to uses annotation.

Notification: Automatically loaded package Complex 3.2.1 due to uses annotation.

Notification: Automatically loaded package ModelicaServices 3.2.1 due to uses annotation.

パッケージは、パスの最初の識別子を調べることによってもロードされます。

```
>>> instantiateModel(Modelica.Electrical.Analog.Basic.Ground)
class Modelica.Electrical.Analog.Basic.Ground "Ground node"
  Real p.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
  Real p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the pin
↪";
equation
  p.v = 0.0;
  p.i = 0.0;
end Modelica.Electrical.Analog.Basic.Ground;
```

注釈:

Notification: Automatically loaded package Complex 3.2.3 due to uses annotation.

Notification: Automatically loaded package ModelicaServices 3.2.3 due to uses annotation.

Notification: Automatically loaded package Modelica default due to uses annotation.

1.2.18 モデルのクエリや操作 API の呼び出し

OpenModelica システムドキュメントには、モデルに関する情報を返したり、モデルの操作を可能にする外部 API（アプリケーションプログラミングインターフェイス）が記載されています。 これらの関数の呼び出しは、以下のようにインタラクティブに実行できますが、より一般的には、OpenModelica コンパイラ（OMC）サーバーに対してクライアントから実行されます。 このようなクライアントの例は、OpenModelica MDT Eclipse プラグイン、OMNotebook、OMEdit グラフィックモデルエディターなどです。この API は、パフォーマンス上の理由から型指定されていません。つまり、型チェックがなく、呼び出しで最小限のエラーチェックが行われます。 呼び出しの結果は、クライアントが解析する必要のある Modelica 構文形式のテキスト文字列として返されます。 C++のパーサーの例は OMNotebook ソースコードで利用できますが、Java の別のパーサーの例は MDTEclipse プラグインで利用できます。

以下に、以前にシミュレートした BouncingBall モデルに対するいくつかの呼び出しコマンドを示します。この API に関する正式なドキュメントは、システムドキュメントで入手できます。 まず、モデルを再度ロードしてリスト表示し、その構造を示します。

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/
↪BouncingBall.mo");
>>> list(BouncingBall)
model BouncingBall
  parameter Real e = 0.7 "coefficient of restitution";
```

(次のページに続く)

(前のページからの続き)

```
  parameter Real g = 9.81 "gravity acceleration";
  Real h(fixed = true, start = 1) "height of ball";
  Real v(fixed = true) "velocity of ball";
  Boolean flying(fixed = true, start = true) "true, if ball is flying";
  Boolean impact;
  Real v_new(fixed = true);
  Integer foo;
equation
  impact = h <= 0.0;
  foo = if impact then 1 else 2;
  der(v) = if flying then -g else 0;
  der(h) = v;
  when {h <= 0.0 and v <= 0.0, impact} then
    v_new = if edge(impact) then -e * pre(v) else 0;
    flying = v_new > 0;
    reinit(v, v_new);
  end when;
end BouncingBall;
```

さまざまな種類の呼び出しとその結果：

```
>>> getClassRestriction(BouncingBall)
"model"
>>> getClassInformation(BouncingBall)
("model","",false,false,false,"/home/OpenModelica/build/share/doc/omc/testmodels/
↪BouncingBall.mo",false,1,1,23,17,{},false,false,"","",false,"")
>>> isFunction(BouncingBall)
false
>>> existClass(BouncingBall)
true
>>> getComponents(BouncingBall)
{{Real,e,"coefficient of restitution", "public", false, false, false, false,
↪"parameter", "none", "unspecified",{}},{Real,g,"gravity acceleration", "public",␣
↪false, false, false, false, "parameter", "none", "unspecified",{}},{Real,h,
↪"height of ball", "public", false, false, false, false, "unspecified", "none",
↪"unspecified",{}},{Real,v,"velocity of ball", "public", false, false, false,
 false, "unspecified", "none", "unspecified",{}},{Boolean,flying,"true, if ball is␣
↪flying", "public", false, false, false, false, "unspecified", "none",
↪"unspecified",{}},{Boolean,impact,"", "public", false, false, false, false,
↪"unspecified", "none", "unspecified",{}},{Real,v_new,"", "public", false, false,
 false, false, "unspecified", "none", "unspecified",{}},{Integer,foo,"", "public",␣
↪false, false, false, false, "unspecified", "none", "unspecified",{}}}
>>> getConnectionCount(BouncingBall)
0
>>> getInheritanceCount(BouncingBall)
0
>>> getComponentModifierValue(BouncingBall,e)
"0.7"
>>> getComponentModifierNames(BouncingBall,"e")
{}
>>> getClassRestriction(BouncingBall)
"model"
>>> getVersion() // Version of the currently running OMC
"OMCompiler v1.14.1"
```

1.2.19 OpenModelica の終了

OpenModelica を終了する

```
>>> quit()
```

1.2.20 XML 形式でのダンプ

コマンド dumpXMLDAE は、いくつかのオプションのパラメータに応じて、モデルの XML をダンプ (出力) します。

dumpXMLDAE(*modelname*[,asInSimulationCode=<Boolean>] [,filePrefix=<String>] [,storeInTemp=<Boolean>] [,addMathMLCode =<Boolean>])

このコマンドは、オプションのパラメーターを使用して、XML を使用してモデルの数学的表現をダンプします。 特に、asInSimulationCode は、変換プロセスのどこで停止するか（モデルをダンプする前）を定義します。他のオプションは、ファイルストレージに関連しています。別の名前を指定するための filePrefix と、一時ディレクトリを使用するための storeInTemp です。 オプションのパラメーター addMathMLCode を使用すると、xml ファイル内に MathML コードを出力せずに、読みやすくすることができます。 使用法は簡単です。addMathMLCode=*true/false* (デフォルトは false).

1.2.21 Matlab 形式でのダンプ

コマンド export は、いくつかのオプションのパラメータに応じて、モデルの XML をダンプします。

"exportDAEtoMatlab(*modelname*);"

このコマンドは、Matlab 形式でモデルの数学的表現をダンプします。

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/
↪BouncingBall.mo")
true
>>> exportDAEtoMatlab(BouncingBall)
"The equation system was dumped to Matlab file:BouncingBall_imatrix.m"
```

```
% Incidence Matrix
% ====================================
% number of rows: 6
IM={{3,6},{1,{'if', 'true','==' {3},{},}},{{'if', 'true','==' {4},{},}},{5},{2,{'if
↪', 'edge(impact)' {3},{5},}},{4,2}};
VL = {'foo','v_new','impact','flying','v','h'};


EqStr = {'impact = h <= 0.0;','foo = if impact then 1 else 2;','der(v) = if flying␣
↪then -g else 0.0;','der(h) = v;','when {h <= 0.0 and v <= 0.0, impact} then v_
↪new = if edge(impact) then (-e) * pre(v) else 0.0; end when;','when {h <= 0.0␣
↪and v <= 0.0, impact} then flying = v_new > 0.0; end when;'};


OldEqStr={'class BouncingBall','  parameter Real e = 0.7 "coefficient of␣
↪restitution";','  parameter Real g = 9.81 "gravity acceleration";','  Real␣
↪h(start = 1.0, fixed = true) "height of ball";','  Real v(fixed = true)
↪"velocity of ball";','  Boolean flying(start = true, fixed = true) "true if␣
↪ball is flying";','  Boolean impact;','  Real v_new(fixed = true);','  Integer␣
↪foo;','equation','  impact = h <= 0.0;','  foo = if impact then 1 else 2;','␣
↪der(v) = if flying then -g else 0.0;','  der(h) = v;','  when {h <= 0.0 and v <=␣
↪0.0, impact} then','    v_new = if edge(impact) then (-e) * pre(v) else 0.0;','
  flying = v_new > 0.0;','    reinit(v, v_new);','  end when;','end BouncingBall;',
↪''};
```

(次のページに続く)

(前のページからの続き)

1.3 インタラクティブセッションハンドラに対するコマンドの概要

以下は、インタラクティブセッションハンドラで現在使用可能なコマンドの一覧です。

simulate(*modelname*)　モデル名 *modelname* を変換しシミュレートします。

simulate(*modelname*[,startTime=<*Real*>][,stopTime=<*Real*>][,numberOfIntervals
"=<*Integer*>][,outputInterval=<*Real*>][,method=<*String*>]"
"[,tolerance=<*Real*>][,fixedStepSize=<*Real*>]"
[,outputFormat=<*String*>]) オプションの開始時間、終了時間、およびオプションの計算回数または間隔を使用してモデルを変換およびシミュレーションします。 計算回数が大きいほど時間分解能は高くなりますが、より多くのスペースを占有し、計算に時間がかかります。 デフォルトの計算回数は 500 です。ソルバーを選択でき、デフォルトは"dassl"で、"euler"、"rungekutta"も使用できます。 出力フォーマット「mat」がデフォルトです。 「plt」と「mat」（MATLAB）は val（）コマンドが有効で、「csv」（コンマ区切り値）と「empty」（出力なし）も使用できます。 **シミュレーション出力形式の変更** を参照してください。

plot(*vars*) ベクトルまたはスカラーとして与えられた変数をプロットします。例. plot({x1,x2}) あるいは plot(x1).

plotParametric(*var1*, *var2*) 最後にシミュレートされたモデルの var1 を基準にして var2 をプロットします。例. plotParametric(x,y).

cd() 現在のディレクトリを返します。

cd(*dir*) ディレクトリを変更します。*dir* は文字列です。

clear() すべての読込済みの定義をクリアします。

clearVariables() すべての変数の定義をクリアします。

dumpXMLDAE(*modelname*, ...) いくつかのオプションのパラメータに応じて、モデルの XML をダンプします。

exportDAEtoMatlab(*name*) モデルの Matlab 表現をダンプします。

instantiateModel(*modelname*) モデル/クラスのインスタンス化の実行とフラット化したクラス定義を文字列として返します。

list() ロードされたすべてのクラス定義を文字列として返します。

list(*modelname*) 指定されたクラスのクラス定義を文字列として返します。

listVariables() 定義されている変数の名前をベクトルで返します。

loadModel(*classname*) 環境変数 OPENMODELICALIBRARY で定義されたパスから *classname* で指定されたモデルやパッケージを読み込みます。

loadFile(*str*) 文字列引数 *str* で指定された Modeica ファイル (.mo) を読み込みます。

readFile(*str*) 文字列引数 *str* で指定されたファイルを読み込み、ファイルの内容を文字列として返します。

runScript(*str*) 文字列引数 *str* で指定されたファイル名のスクリプトファイルを実行します。

system(*str*) オペレーティング・システム内のシステム（シェル）コマンドとして *str* を実行します。コマンドの成否を整数値で返します。シェルコマンドから標準出力へ出力された結果はコンソールウィンドウに返されます。

timing(*expr*) 式 *expr* を評価し、かかった秒数（経過時間）を返します。

typeOf(*variable*) 文字列として変数 *variable* の型を返します。

saveModel(*str*,*modelname*) 文字列引数 *str* で指定したファイルに、指定された名前 *modelname* でモデル/クラスを保存します。

val(*variable*,*timePoint*) 時間 *timePoint* 時点の変数 *variable* の補間値を返します。

help() 文字列としてヘルプテキストを返します。

quit() OpenModelica 環境を終了します。

1.4 コマンドラインからコンパイラを実行する

OpenModelica コンパイラは、Windows の cmd.exe でコマンドラインから使用することができます。

例. セッション 1 - コマンドラインパラメータに関する情報を取得

```
C:\dev> C:\OpenModelica1.9.2 \bin\omc -h
OpenModelica Compiler 1.9.2
Copyright 2015 Open Source Modelica Consortium (OSMC)
Distributed under OMSC-PL and GPL, see https://www.openmodelica.org/
Usage: omc [Options] (Model.mo | Script.mos) [Libraries | .mo-files]
...
```

例. セッション 2 - TestModel.mo ファイルを作成し omc で実行する

```
C:\dev> echo model TestModel parameter Real x = 1; end TestModel; > TestModel.mo
C:\dev> C:\OpenModelica1.9.2 \bin\omc TestModel.mo
class TestModel
      parameter Real x = 1.0;
end TestModel;
C:\dev>
```

例. セッション 3 - scrypt.mos ファイルを作成し omc で実行する

ご使用のエディタで script.mos ファイルを作成し以下のコマンドを記述してください。

```
// start script.mos
loadModel(Modelica); getErrorString();
simulate(Modelica.Mechanics.MultiBody.Examples.Elementary.Pendulum); getErrorString();
// end script.mos
C:\dev> notepad script.mos
C:\dev> C:\OpenModelica1.9.2 \bin\omc script.mos
"true"
""
record SimulationResult
    resultFile = C:/dev/Modelica.Mechanics.MultiBody.Examples.Elementary.Pendulum_res.mat","
    simulationOptions = startTime = 0.0, stopTime = 5.0, numberOfIntervals "= 500, tolerance = 1e-006,
    method = 'dassl', fileNamePrefix = 'Modelica.Mechanics.MultiBody.Examples.Elementary.Pendulum',
    options = '', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ","
    messages = "",
    timeFrontend = 1.245787339209033,
    timeBackend = 20.51007138993843,
    timeSimCode = 0.1510248469321959,
    timeTemplates = 0.5052317333954395,
    timeCompile = 5.128213942691722,
    timeSimulation = 0.4049189573103951,
    timeTotal = 27.9458487395605
end SimulationResult;
""
```

コンパイラからより多くの情報を取得するには、コンパイラの実行時にコマンドラインオプション **--showErrorMessages -d=failtrace** を使用してください。

```
C:\dev> C:\OpenModelica1.9.2 \bin\omc --showErrorMessages -d=failtrace script.mos
```

第2章　OMEdit - OpenModelica Connection Editor

OMEdit - OpenModelica Connection Editor は、OpenModelica でグラフィカルにモデルを編集するための新しいグラフィカルユーザーインターフェイスです。 Qt グラフィカルユーザーインターフェイスライブラリを使用して C ++で実装されており、最新の OpenModelica のインストーラに含まれている Modelica Standard Library をサポートします。 この章では、OMEdit の概要を簡単に説明し、エディターを使用して DCMotor モデルを作成する方法も示します。(訳注：OMEdit には英語環境や日本語環境がありますが、本書では画面のキャプチャ画像は英語版を使用しております。本書内のメニューの説明には英語と日本語を併記しています。)

OMEdit はモデルの作成、閲覧、編集およびシミュレートするためのいくつかのユーザーフレンドリーな機能を提供しています。

- **モデリング** - Modelica モデルの簡単な作成。
- **事前定義モデル** - Modelica Standard Library をブラウジングしモデルにアクセスできます。
- **ユーザー定義モデル** - ユーザーはすぐに使用でき後で再利用できる独自のモデルを作成できます。
- **コンポーネントインターフェイス** - モデルインターフェイス間の接続を描画し、編集するためのスマートな接続編集機能です。
- **シミュレーション** - 計算開始時のシミュレーション設定や開始時間や終了時間などを指定しシミュレーションを実行するサブシステムです。
- **プロッティング** - シミュレートされたモデルから変数をプロットするインターフェイスです。

2.1 OMEdit の開始

OMEdit を起動すると Figure 2.1 に示すようなスプラッシュ画面が表示されます。 実行ファイルはプラットフォームに応じてさまざまな場所にあります（以下を参照)。

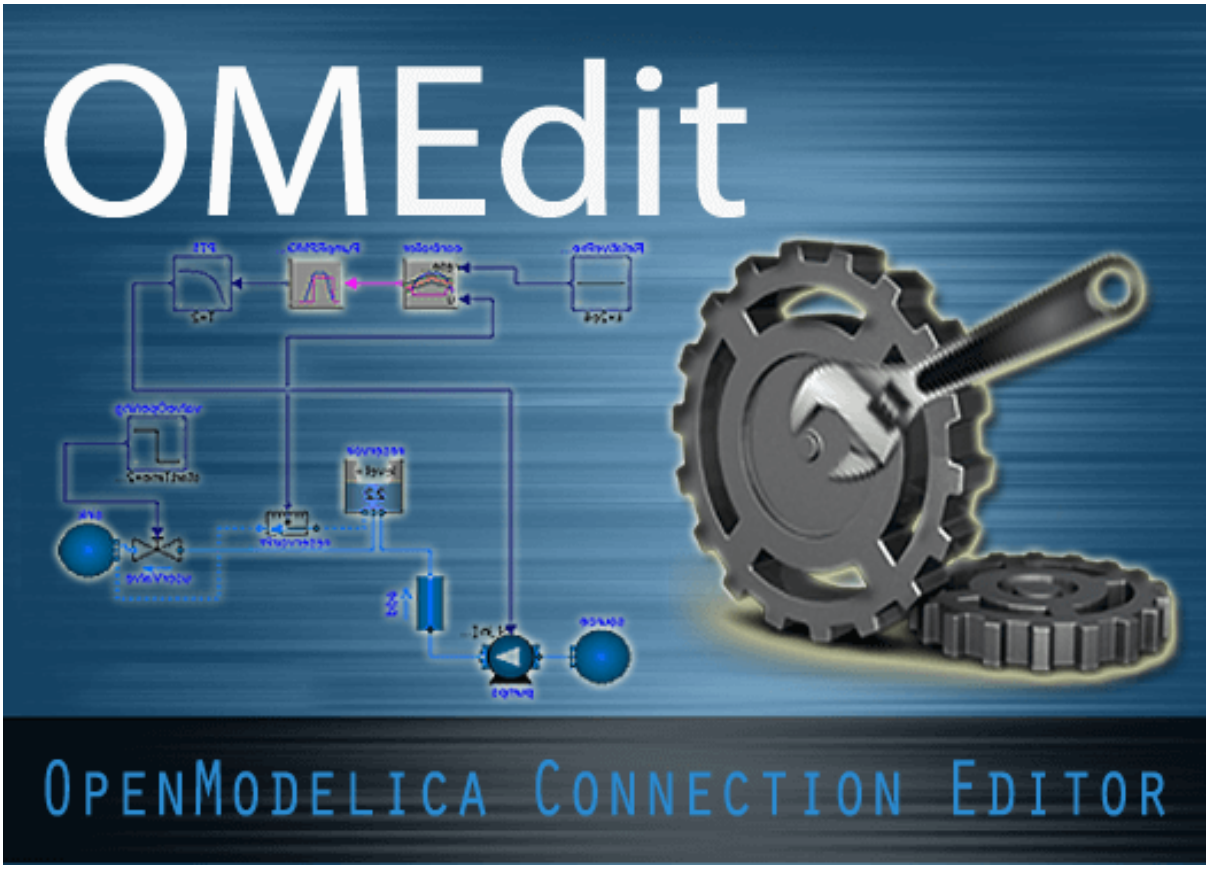


Figure 2.1: OMEdit スプラッシュ画面

2.1.1 Microsoft Windows

OMEdit は実行ファイル OpenModelicaInstallationDirectory/bin/OMEdit/OMEdit.exe から起動することができます。または、Windows の [スタート] メニューから OpenModelica> OpenModelica Connection Editor を選択して起動することも出来ます。

2.1.2 Linux

対応するメニューアプリケーション項目を選択するか、シェルまたはコマンドプロンプトで "OMEdit" と入力して OMEdit を開始します。

2.1.3 Mac OS X

デフォルトのインストールは/Application/MacPorts/OMEdit.app です。

2.2 メインウィンドウ＆ブラウザ

メインウィンドウには、いくつかのドッキング可能なブラウザが含まれています、

- Libraries Browser(ライブラリブラウザ)
- Documentation Browser(ドキュメントブラウザ)
- Variables Browser(変数ブラウザ)
- Messages Browser(メッセージブラウザ)

Figure 2.2 にメインウィンドウとブラウザを示します。

ブラウザのデフォルトの配置は Figure 2.2 に示されています。 メッセージブラウザを除くすべてのブラウザは、左または右の列にドッキングできます。 メッセージブラウザは、上部または下部の領域にドッキングできます。 OMEdit にブラウザの新しいドッキング位置を記憶させたい場合は、[ユーザーの GUI カスタマイズの保持] オプションを有効にする必要があります。セクション *General(全般)* を参照してください。

2.2.1 クラスのフィルタ

クラスをフィルタリングするには、[編集]> [Filter Classes] をクリックするか、キーボードショートカットの Ctrl + Shift + F を押します。 ロードされた Modelica クラスは、クラス名の任意の部分を入力することでフィルタリングできます。

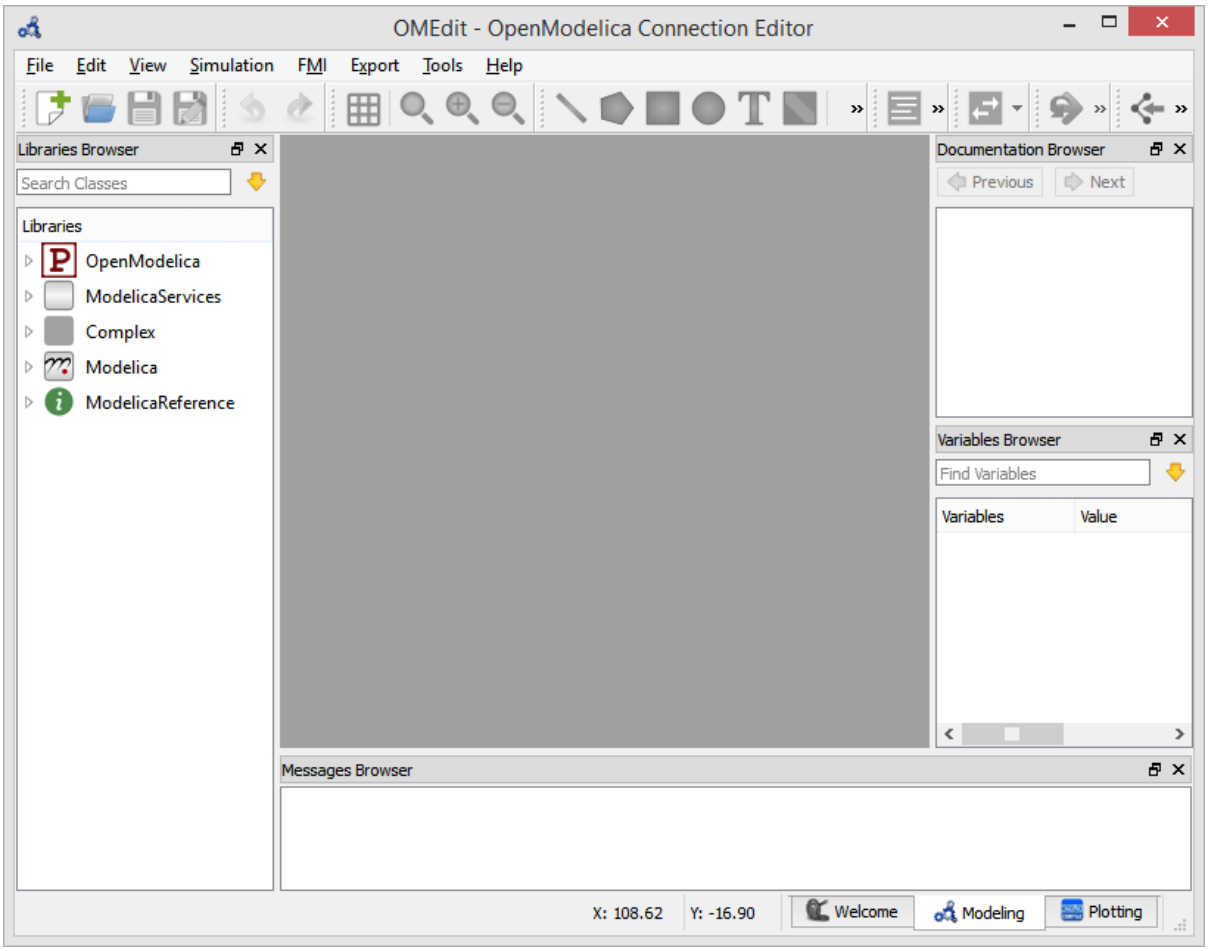


Figure 2.2: OMEdit メインウィンドウとブラウザ

2.2.2 Libraries Browser(ライブラリブラウザ)

ライブラリブラウザを表示するには、[ビュー]> [ウィンドウ]> [ライブラリブラウザ] をクリックします。 ライブラリブラウザはロードされた Modelica クラスのリストを表示します。 ライブラリブラウザの各項目には、クラスの操作と使用を簡単にするための右クリックメニューがあります。 クラスは、名前とアイコンが付いたツリー構造で表示されます。 保護されたクラスはデフォルトでは表示されません。 保護されたクラスを表示する場合は、[プロテクトされているクラスの表示] オプションを有効にする必要があります。セクション *General(全般)* を参照してください。

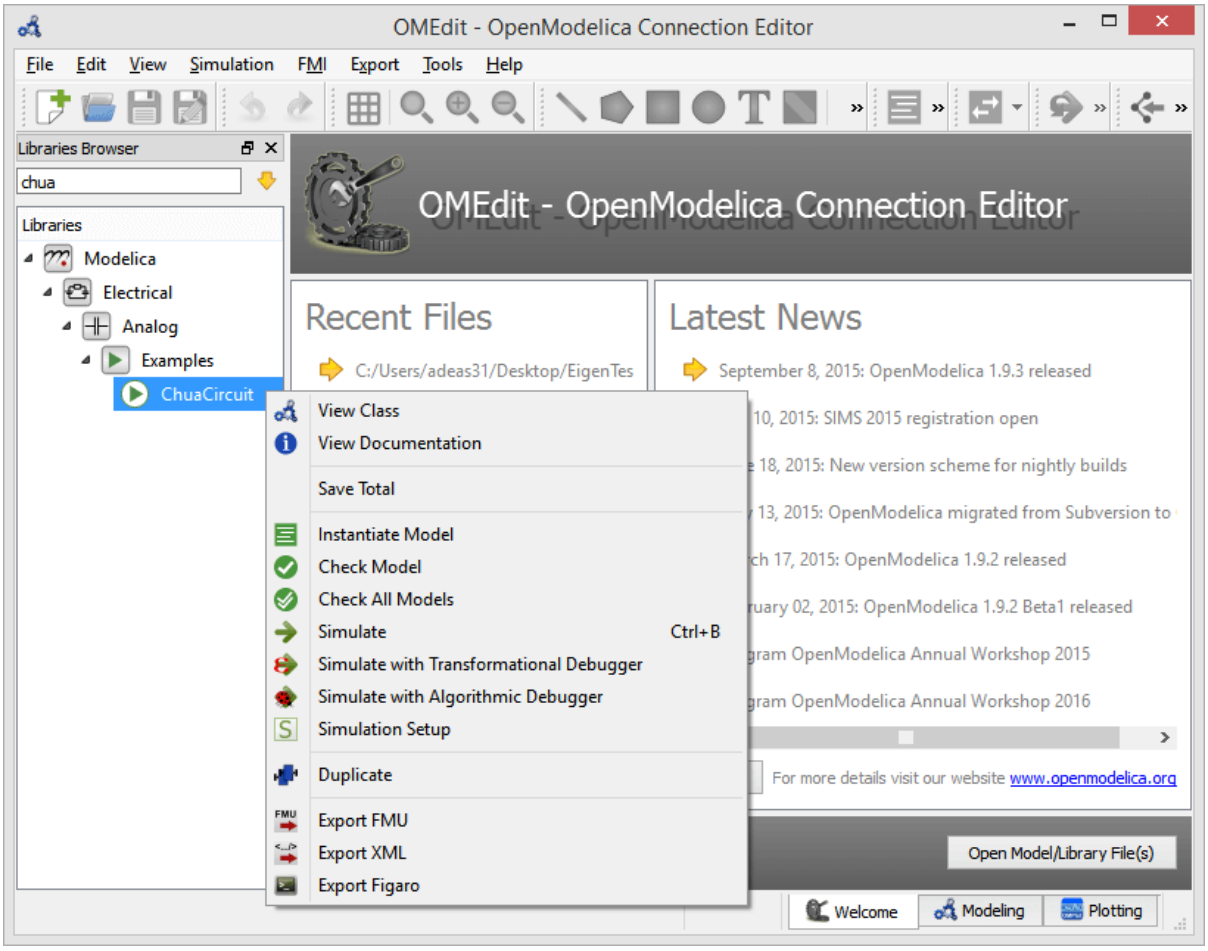


Figure 2.3: Libraries Browser(ライブラリブラウザ)

2.2.3 Documentation Browser(ドキュメントブラウザ)

Modelica クラスの HTML ドキュメントを表示します。 ページの前後に移動するためのナビゲーションボタンが含まれています。 また、HTML 形式でクラスドキュメントを作成できる WYSIWYG エディタも含まれています。 クラスのドキュメントを表示するには、ライブラリブラウザで Modelica クラスを開き、[ドキュメントビュー] を選択します。

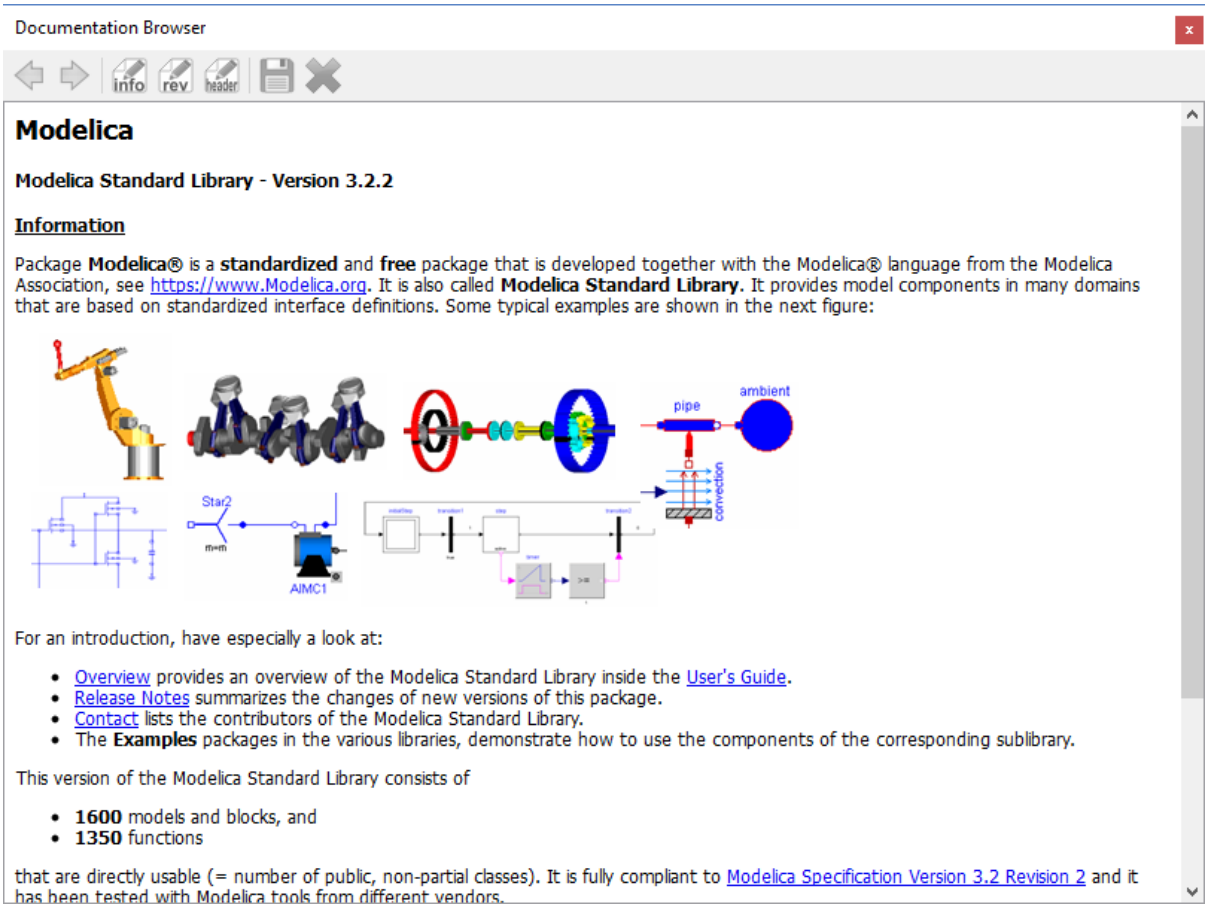


Figure 2.4: Documentation Browser(ドキュメントブラウザ)

2.2.4 Variables Browser(変数ブラウザ)

クラスの変数はツリー形式で構造化され、変数ブラウザーに表示されます。 各変数にはチェックボックスがあります。 チェックボックスをオンにすると、変数値がプロットされます。 ツリー内の変数をフィルタリングするための検索ボックスが上部にあります。 フィルタリングは、正規表現、ワイルドカードおよび固定文字列を使用して実行できます。 変数ブラウザは、[Collapse All(すべて折りたたむ)] ボタンと [Expand All(すべて展開)] ボタンを使用して折りたたんだり展開したりできます。

ブラウザでは、 再シミュレートするモデル で解説されているように変更可能なパラメータの操作が可能です。 また、変数の単位と説明も表示されます。

ブラウザには、スライダーとアニメーションボタンも含まれています。 これらのコントロールは、モデルの可変グラフィックスおよびスケマティックアニメーション、つまり DynamicSelect アノテーションに使用されます。 また、ステートマシンのデバッグにも使用されます。 アニメーション用に ダイアグラムウィンドウ を開きます。 一度に描画できるのは 1 つのモデルのみです。 これは、変数ブラウザで結果ファイルをアクティブにすることで実現されます。 アニメーションは、アクティブな結果ファイルから値を読み取るだけです。 複数のモデルをシミュレートすることが可能です。 その場合、ユーザーには変数ブラウザーに結果ファイルのリストが表示されます。 ユーザーは、結果ファイルを右クリックし、コンテキストメニューで [Set Active] を選択することで、異なる結果ファイルを切り替えることができます。

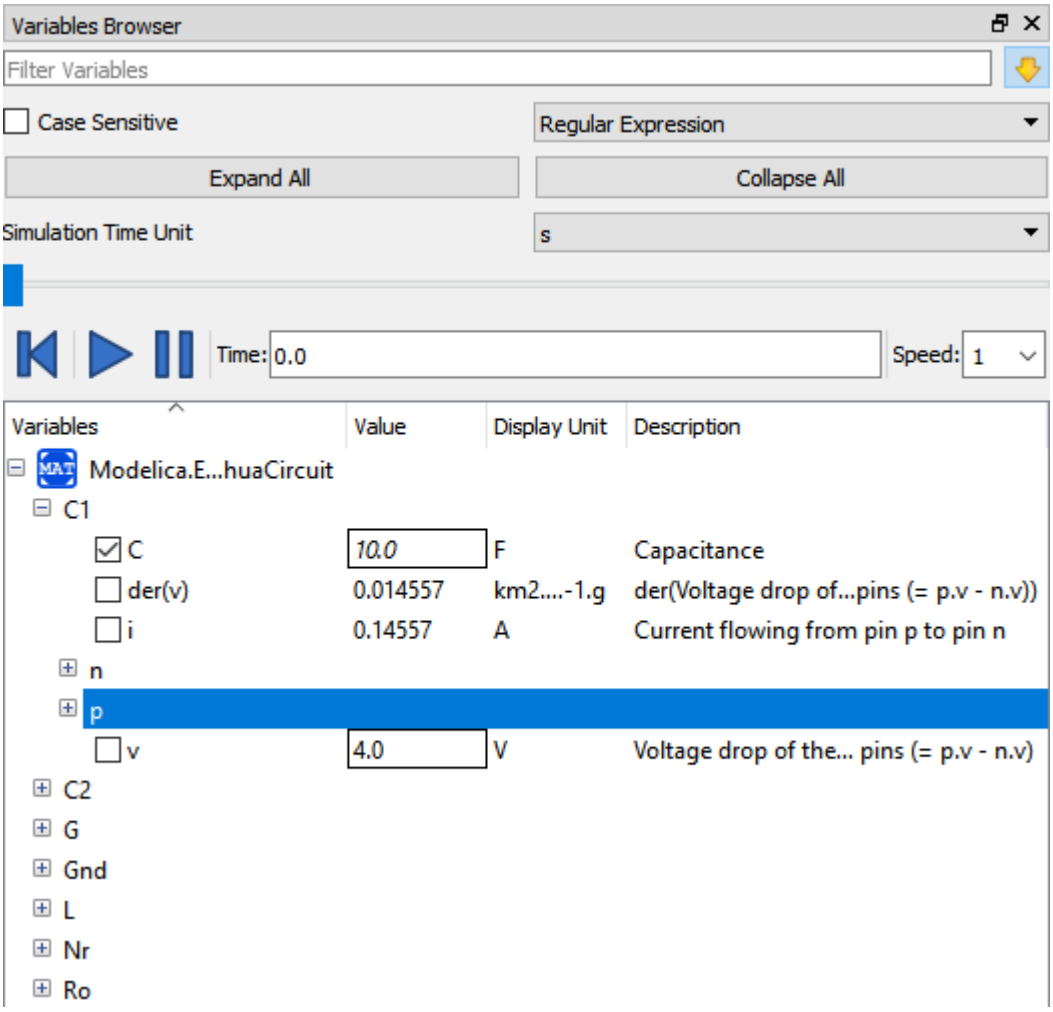


Figure 2.5: Variables Browser(変数ブラウザ)

2.2.5 Messages Browser(メッセージブラウザ)

エラーのリストを表示します。 次のような種類のエラーが発生する可能性があります。

- 構文
- 文法
- 翻訳
- シンボリック
- シミュレーション
- スクリプティング

メッセージブラウザのオプションについては、セクション *Messages(メッセージ)* を参照してください。

2.3 パースペクティブ

パースペクティブタブは、メインウィンドウの右下にあります。

- Welcome(ようこそ) パースペクティブ
- Modeling(モデリング) パースペクティブ
- Plotting(プロット) パースペクティブ
- Debugging パースペクティブ

2.3.1 Welcome(ようこそ) パースペクティブ

ようこそパースペクティブには、最近のファイルのリストと https：//www.openmodelica.org/から取得した最新ニュースのリストが表示されます。 Figure 2.6 を参照してください。 最近のファイルと最新ニュースの向きは、水平方向または垂直方向にすることができます。 ユーザーは最新ニュースを表示/非表示にすることができます。 セクション *General(全般)* を参照してください。

2.3.2 Modeling(モデリング) パースペクティブ

モデリングパースペクティブは、ユーザーがモデルを作成および設計できるインターフェイスを提供します。 Figure 2.7 を参照してください。

モデリングパースペクティブインターフェイスは、タブ付きビューとサブウィンドウビューの 2 つの異なるモードで表示できます。セクション *General(全般)* を参照してください。

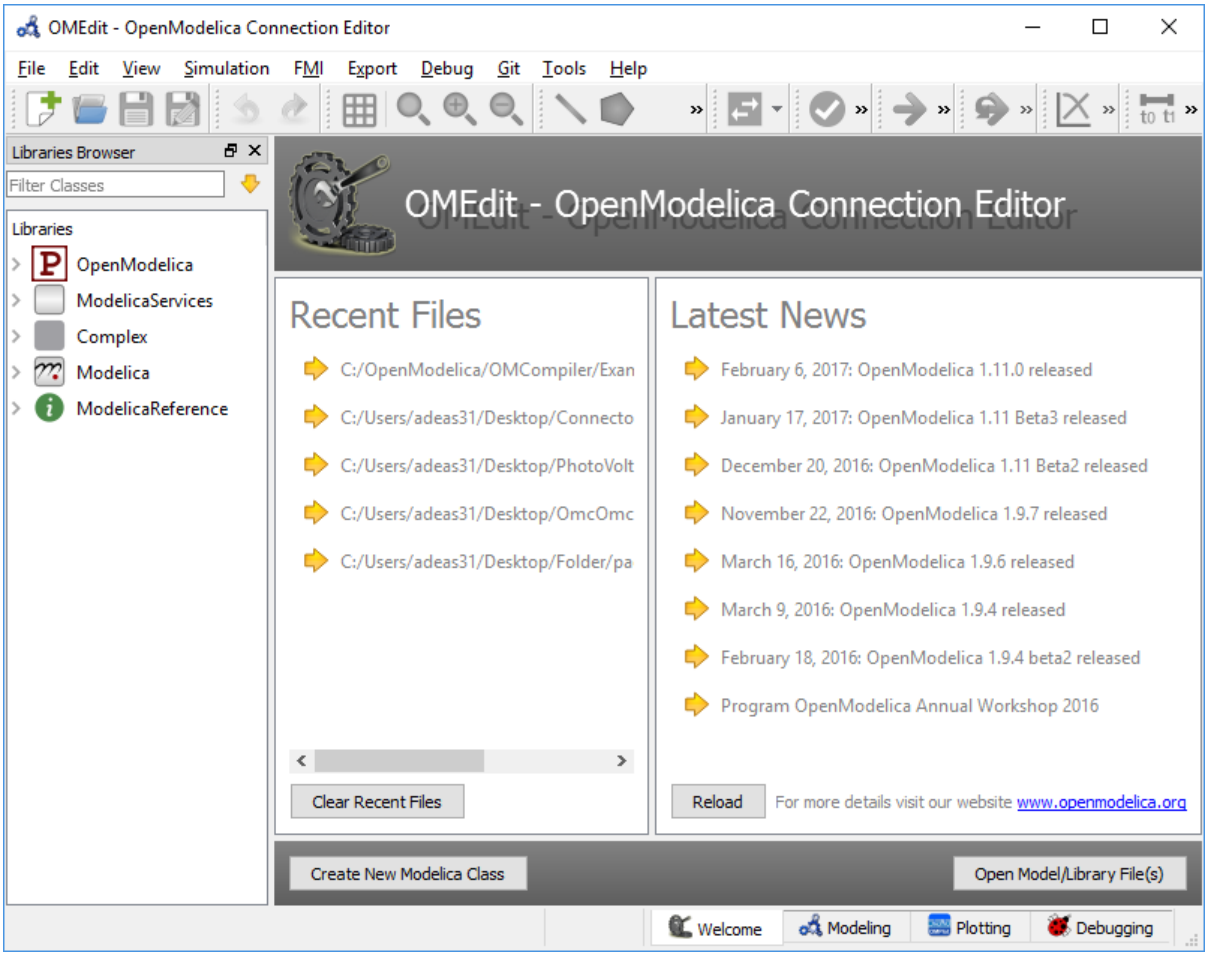


Figure 2.6: OMEdit ウェルカムパースペクティブ

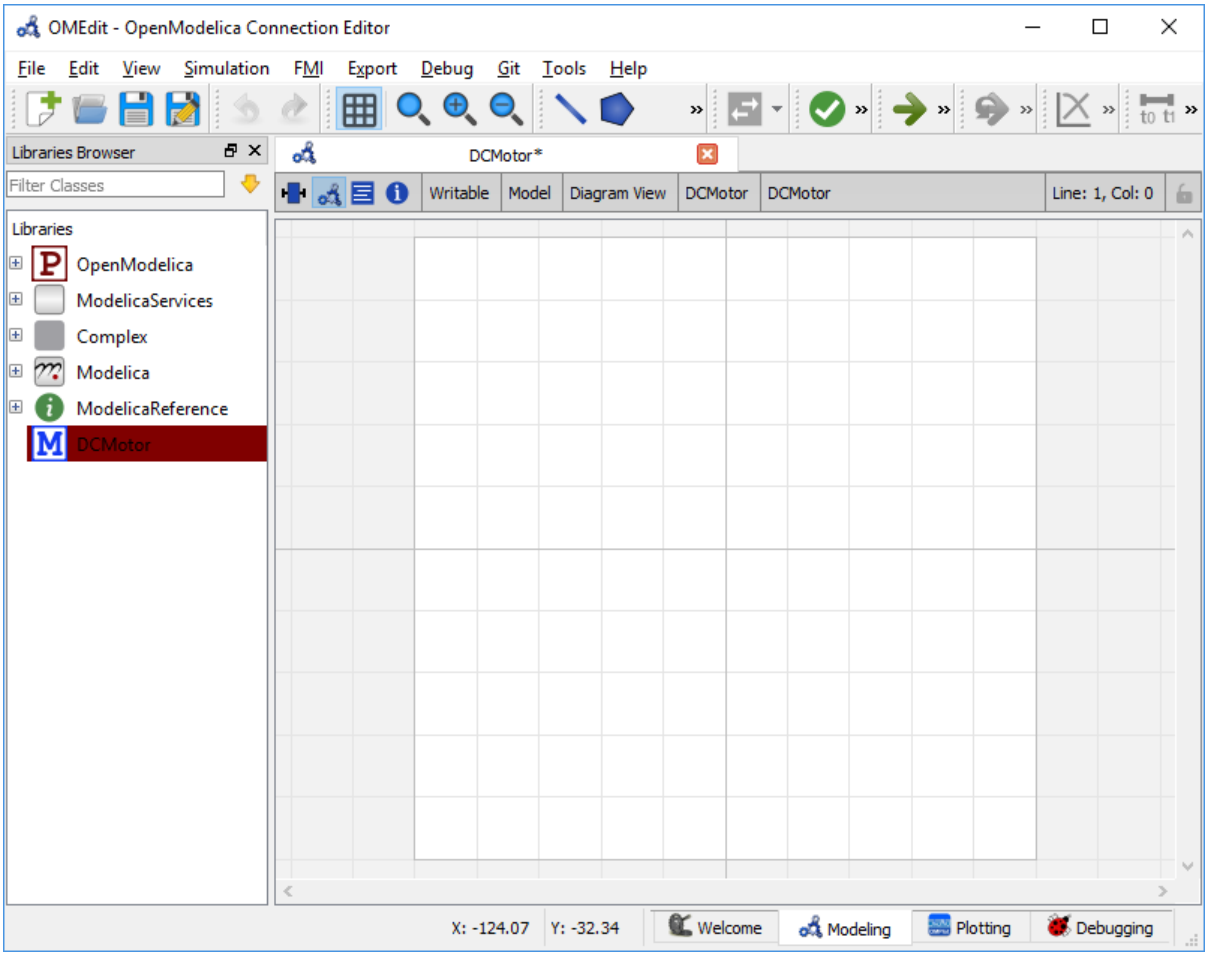


Figure 2.7: OMEdit モデリングのパースペクティブ

2.3.3 Plotting(プロット) パースペクティブ

プロットパースペクティブは、モデルのシミュレーション結果を示します。 モデルのシミュレーションが正常に終了すると、プロットパースペクティブが自動的にアクティブになります。 また、ユーザーがOpenModelica でサポートされている結果ファイルのいずれかを開いたときにもアクティブになります。 モデリングパースペクティブと同様に、このパースペクティブは、タブ付きビューとサブウィンドウビューの 2 つの異なるモードで表示することもできます。セクション *General(全般)* を参照してください。

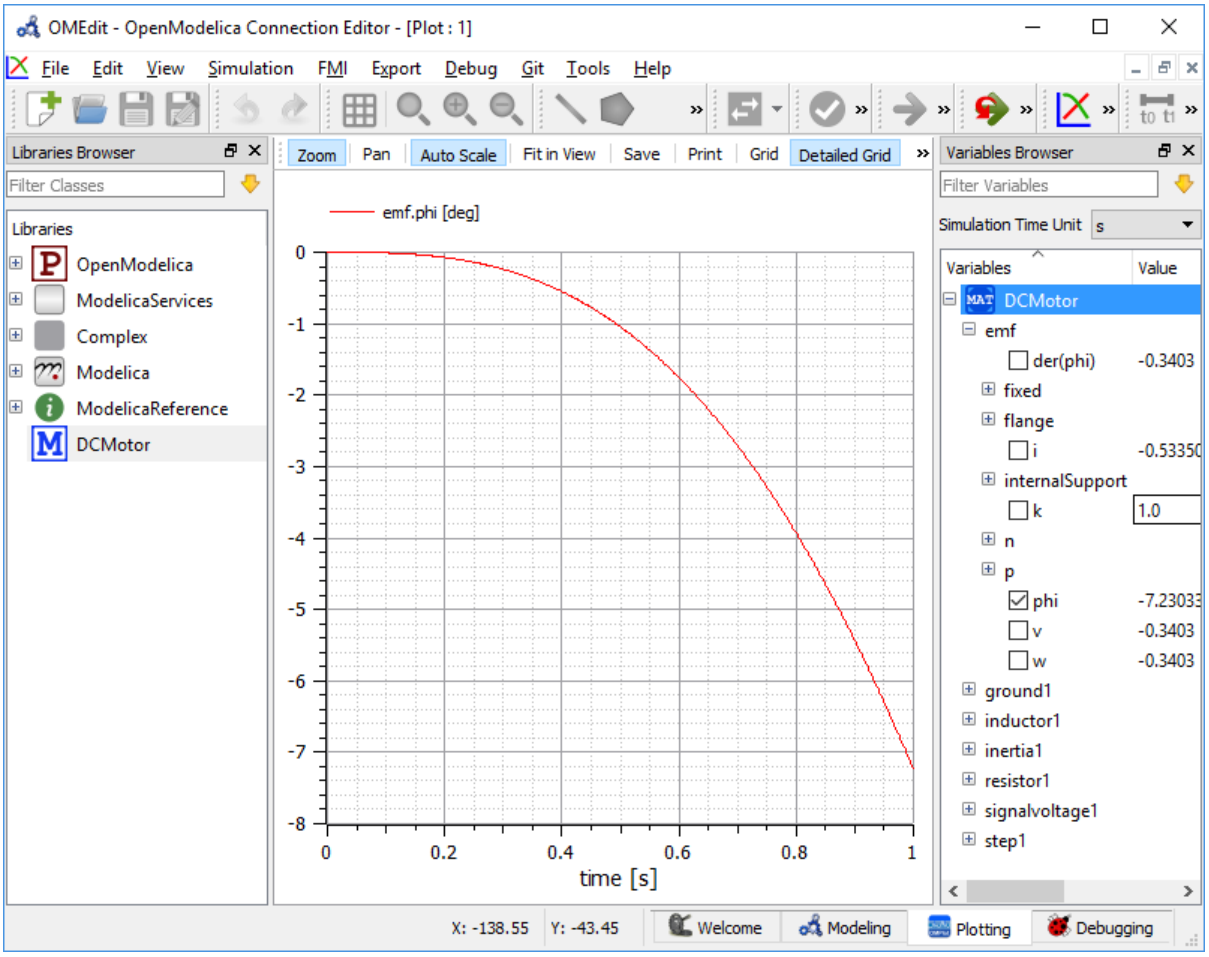


Figure 2.8: OMEdit プロッティングパースペクティブ

2.3.4 Debugging パースペクティブ

ユーザーがアルゴリズムデバッガーを使用してクラスをシミュレートすると、アプリケーションは自動的にデバッグパースペクティブに切り替わります。 パースペクティブには、スタックフレーム、ブレークポイントおよび変数のリストが表示されます。

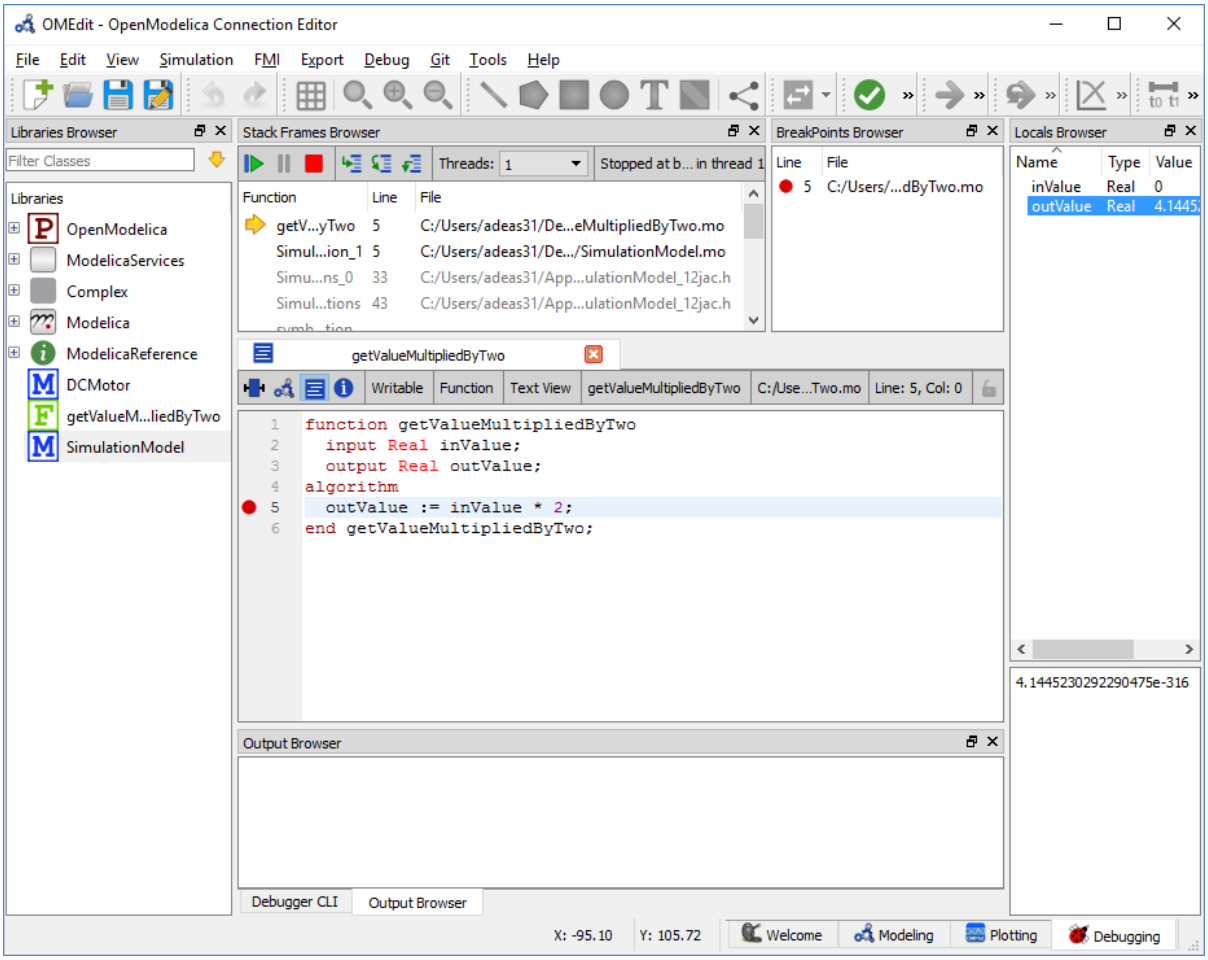


Figure 2.9: OMEdit デバッグパースペクティブ

2.4 File(ファイル) メニュー

- *New Modelica Class(Modelica* **クラス新規作成***)* - 新しい Modelica クラスを作成します。
- *Open Model/Library File(s)(***モデル/ライブラリを開く***)* - Modelica ファイルやライブラリを開く。
- *Open/Convert Modelica File(s) With Encoding(***エンコードを用いて** *Modelica* **ファイルを開く/変換する***)* - Modelica ファイルまたは特定のエンコーディングを使用するライブラリを開きます。 UTF-8 に変換することも可能です。
- *Load Library(***ライブラリのロード***)* - Modelica ライブラリを読み込みます。 パスに package.mo ファイルが含まれているとして、ユーザーがライブラリパスを選択できるようにします。
- *Load Encrypted Library* - 暗号化ライブラリをロードします。 *OpenModelica Encryption* を参照してください
- *Open Result File(s)(***結果ファイルを開く***)* - 結果ファイルを開きます。
- *Open Transformations File(***変換ファイルを開く***)* - トランスフォーメーションデバッガファイルを開きます。
- *New Composite Model* - 新しいコンポジットモデルを作成します。
- *Open Composite Model(s)* - コンポジットファイルを開きます。
- *Load External Model(s)(***外部のモデル** *(***複数***)* **をロード***)* - コンポジットモデル内で使用できる外部モデルをロードします。
- *Open Directory* - 再帰的にディレクトリのファイルをロードします。ファイルはテキストファイルとしてロードされます。
- *Save(***保存***)* - クラスを保存します。
- *Save As(***別名で保存***)* - クラスとして名前を付けて保存します。
- *Save Total(***すべて保存***)* - クラスとそれが使用するすべてのクラスを 1 つのファイルに保存します。 クラスとその依存関係は、スクリプトで *loadFile()* API 関数を使用してロードできます。 ライブラリの依存関係を気にすることなく、サードパーティがクラスの内容を再現できるようにします。
- *Import*
- *FMU* - FMU をインポートします.
- *FMU Model Description* - FMU の詳細をインポートします.
- *From OMNotbook* - OMNotebook から Modelica モデルをインポートします。
- *Ngspice netlist* - ngspice netlist を Modelica コードにインポートします。
- "Export"
- *To Clipboard* - 現在のモデルをクリップボードにエクスポートします。
- *Image(***画像***)* - 現在のモデルを画像としてエクスポートします。
- *FMU* - 現在のモデルを FMU にエクスポートします。

- *Read-only Package* - 拡張ファイル.mol を用いて zip 化された Modelica ライブラリをエクスポートします。
- *Encrypted Package* - 暗号化されたパッケージをエクスポートします。 *OpenModelica Encryption* 参照。
- *XML* - 現在のモデルを xml ファイルへエクスポートします。
- *Figaro* - 現在のモデルを Figaro にエクスポートします。
- *To OMNotebook* - 現在のモデルを OMNotebook ファイルにエクスポートします。
- *System Libraries(**システムライブラリ**)* - システムライブラリのリストが含まれています。
- *Recent Files(**最近使ったファイル**)* - 最近使用したファイルのリストが含まれています。
- *Clear Recent Files(**最近使ったファイルのクリア**)* - 最近使用したファイルのリストをクリアします。
- *Print(**印刷**)* - 現在のモデルを印刷します。
- *Quit(**終了**)* - OpenModelica Connection Editor を終了します。

2.5 Edit(編集) メニュー

- *Undo(**元に戻す**)* - 最後の変更を元に戻します。
- *Redo(**繰り返す**)* - 最後に取り消された変更をやり直します。
- *Filter Classes* - ライブラリブラウザ内のクラスをフィルターします。 **クラスのフィルタ** を参照してください。

2.6 View(ビュー) メニュー

- *Toolbars(**ツールバー**)* - ツールバーのトグルの表示/非表示の切り替えます。
- *Windows(**ウィンドウ**)* - ウィンドウの表示/非表示の切り替えます。
- *Close Window(**ウィンドウを閉じる**)* - 現在のモデルのウィンドウを閉じます。
- *Close All Windows(**全てのウィンドウを閉じる**)* - すべてのモデルのウィンドウを閉じます。
- *Close All Windows But This(**このウィンドウを残して他の全て閉じる**)* - 現在のウィンドウを除くすべてのモデルウィンドウを閉じます。
- *Cascade Windows(**ウィンドウのカスケード**)* - カスケードパターンですべてのウィンドウを並べる。
- *Tile Windows Horizontally(**ウィンドウを上下に並べる**)* - すべてのウィンドウを上下方向に並べて表示します。
- *Tile Windows Vertically(**ウィンドウを左右に並べる**)* - すべてのウィンドウを垂直に並べて表示します。
- *Toggle Tab/Sub-window View* - タブビューとサブウィンドウビューを切り替えます。
- *Grid Lines(**グリッド線**)* - 現在のモデルのグリッド線を切り替えます。

- *Reset Zoom(***ズームをリセット***)* - 現在のモデルの拡大/縮小表示をリセットします。
- *Zoom In(***ズームイン***)* - 現在のモデルを拡大表示します。
- *Zoom Out(***ズームアウト***)* - 現在のモデルを縮小表示します。

2.7 Simulation(シミュレーション) メニュー

- *Instantiate Model(***モデルのインスタンス化***)* - 現在開いているモデルをインスタンス化します。
- *Check Model(***モデルチェック***)* - 現在開いているモデルをチェックします。
- *Check All Models(***全てのモデルをチェック***)* - ライブラリのすべてのモデルをチェックします。
- *Simulate(***シミュレート***)* - 現在開いているモデルをシミュレートします。
- *Simulate with Transformational Debugger(***変換デバガでシミュレート***)* - 現在開いているモデルをシミュレートし変換デバッガを開きます。
- *Simulate with Algorithmic Debugger(***アルゴリズムデバガでシミュレート***)* - 現在開いているモデルをシミュレートしアルゴリズムデバッガを開きます。
- *Simulate with Animation* - 現在開いているモデルをシミュレートしアニメーションを開きます。
- *Simulation Setup(***シミュレーションのセットアップ***)* - シミュレーションの設定ウィンドウを開きます。

2.8 Debugger(デバグ) メニュー

- *Debug Configurations(***デバグの設定***)* - デバッグの設定ウィンドウを開きます。
- *Attach to Running Process(***実行プロセスへアタッチ***)* - 実行中のプロセスにアルゴリズムデバッガを追加します。

2.9 OMSimulator メニュー

- *New OMSimulator Model* - 新しい OMSimulator モデルを作成します。
- *Open OMSimulator Model(s)* - OMSimulator モデル (複数) を開きます。
- *Add System* - システムにモデルを追加します。
- *Add/Edit Icon* - システム/サブシステムアイコンを追加/編集します。
- *Delete Icon* - システム/サブモデルアイコンを削除します。
- *Add Connector* - システム/サブモデルにコネクタを追加します。
- *Add Bus* - システム/サブモデルにバスを追加します。
- *Add TLM Bus* - システム/サブモデルに TLM バスを追加します。

- *Add SubModel* - システムにサブシステムを追加します。
- *Instantiate Model(***モデルのインスタンス化***)* - モデルをインスタンス化します。
- *Simulate(***シミュレート***)* - モデルをシミュレートします。
- *Archived Simulations(***アーカイブされたシミュレーション***)* - アーカイブされたシミュレーションウィンドウを開きます。

2.10 Tools(ツール) メニュー

- *OpenModelica Compiler CLI(OpenModelica* **コンパイラ** *CLI)* - OpenModelica コンパイラコマンドラインインターフェイスウィンドウを開きます。
- *OpenModelica Command Prompt(OpenModelica* **コマンドプロンプト***)* - OpenModelica コマンドプロンプトを開きます（Windows のみ使用可能）。
- *Open Working Directory(***作業フォルダを開く***)* - 現在の作業ディレクトリを開きます。
- *Open Terminal(***端末を開く***)* - *General(*全般*)* にセットされたターミナルコマンドを実行します。
- *Options(***オプション***)* - オプションウィンドウを開きます。

2.11 Help(ヘルプ) メニュー

- *OpenModelica Users Guide(OpenModelica* **ユーザーガイド***)* - OpenModelica ユーザーガイドを開きます。
- *OpenModelica Users Guide (PDF)(OpenModelica* **のユーザーガイド（***PDF***）** *)* - OpenModelica のユーザーガイド（PDF）を開きます。
- *OpenModelica System Documentation(OpenModelica* **システムドキュメント***)* - OpenModelica のシステムドキュメントを開きます。
- *OpenModelica Scripting Documentation(OpenModelica* **スクリプトドキュメント***)* - OpenModelica のスクリプトドキュメントを開きます。
- *Modelica Documentation(Modelica* **ドキュメント***)* - Modelica のドキュメントを開きます。
- *OMSimulator Users Guide* - OMSimulator のユーザーガイドを開きます。
- *OpenModelica TLM Simulator Documentation* - OpenModelica TLM シミュレータのドキュメントを開きます。
- *About OMEdit(OMEdit* **について***)* - OpenModelica Connection Editor についての情報を表示します。

2.12 モデルを作成する

2.12.1 Modelica クラスの新規作成

OMEdit で新しい Modelica クラスを作成するのは簡単です。 次のいずれかの方法を選択してください。

- メニューから [ファイル]> [Modelica クラス新規作成] を選択します。
- ツールバーの Modelica クラス新規作成ボタンをクリックします。
- ウェルカムパースペクティブの左下にある Modelica クラス新規作成ボタンをクリックします。
- Ctrl + N をクリックします。

2.12.2 Modelica ファイルを開きます。

次のいずれかの方法を選択して、Modelica ファイルを開きます。

- メニューから [ファイル]> [Modelica/Library ファイルを開く] を選択します。
- ツールバーの Model.Library ファイルボタンをクリックします。
- ウェルカムパースペクティブの右下にある [モデル/ライブラリファイルを開く] ボタンをクリックします。
- Ctrl キー+O をクリックします。

（注意事項, MSL のような Modelica システムファイルを編集する場合（非推奨)、以下を参照してください。 *Modelica Standard Library* **の編集** ）

2.12.3 エンコードして Modelica ファイルを開く

メニューから [ファイル]> [エンコーディングを使用して Modelica ファイルを開く/変換] を選択します。ファイルを UTF-8 に変換することなどが可能です。

2.12.4 モデルウィジェット

Modelica クラスごとに 1 つのモデルウィジェットが作成されます。 モデルウィジェットにはステータスバーとビューエリアがあり、ステータスバーにはビューとインフォメーションに対するラベルの間を移動するためのボタンが含まれています。 ビューエリアは、Modelica クラスのアイコン、ダイアグラム、およびテキストレイヤーを表示するために使用されます。 Figure 2.10 を参照してください。

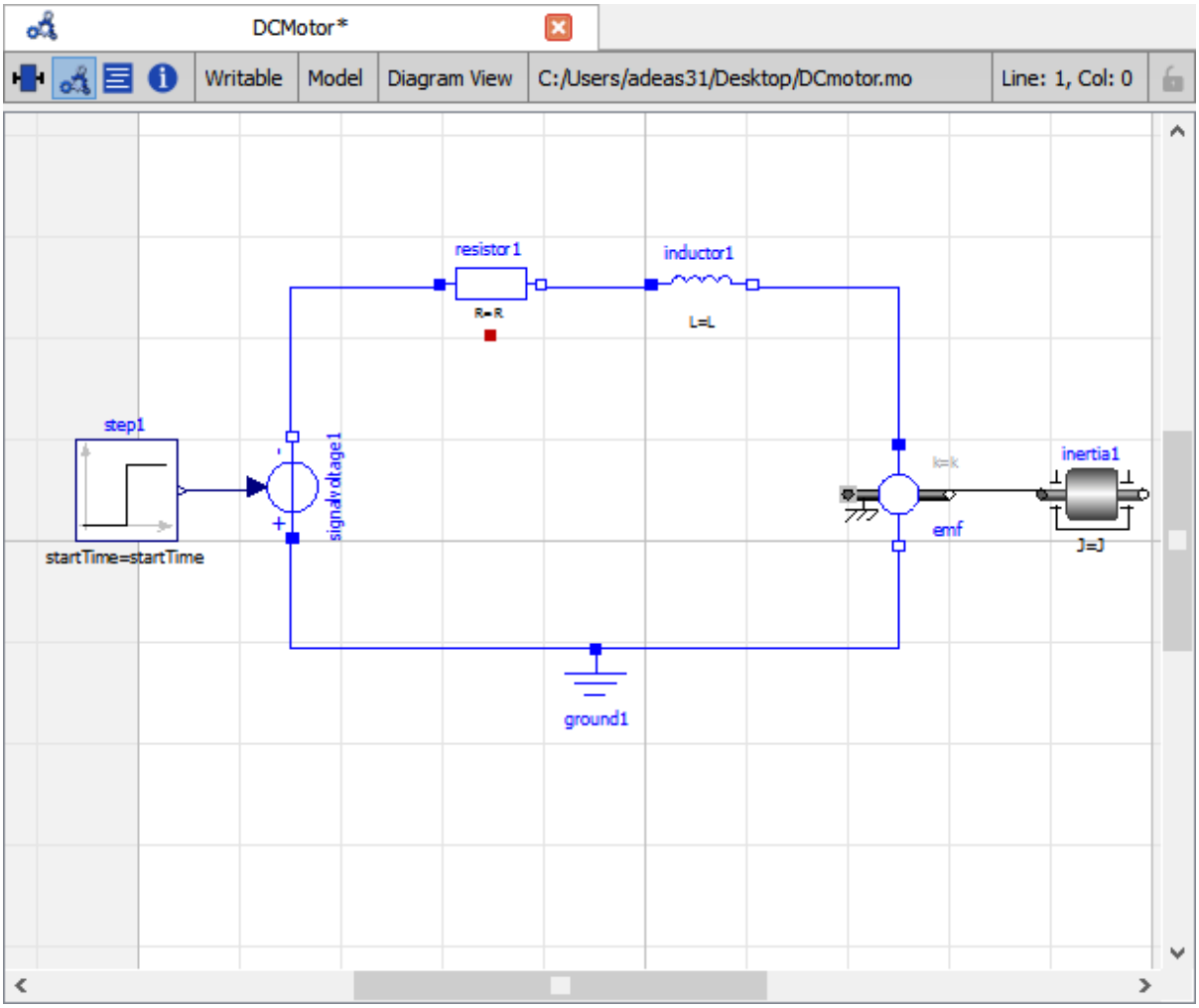


Figure 2.10: ダイアグラムビューを表示しているモデルウィジェット

2.12.5 コンポーネントモデルを追加

ライブラリブラウザからモデルをドラッグし、モデルウィジェットのダイアグラムビューまたはアイコンビューにドロップします。

2.12.6 コネクションの作成

あるコンポーネントモデルを別のコンポーネントモデルに接続するには、ユーザーは最初にツールバーから接続モード () を有効にする必要があります。

コネクタの上にマウスを移動します。すると、 マウスカーソルが矢印カーソルからクロスカーソルに変わります。 接続を開始するには、左ボタンを押し、ボタンを押したまま移動します。 次に、左ボタンを放します。 エンドコネクタに向かって移動し、カーソルがクロスカーソルに変わったらクリックします。

2.13 モデルのシミュレーション

各モデルのシミュレーションオプションは、OMEdit データ構造内に保存されます。 それらのオプションは以下の特徴があります。

- 各モデルには各々シミュレーションオプションがあります。
- モデルが最初に開かれた時は、シミュレーションのオプションがデフォルトに設定されています。
- `experiment` と `__OpenModelica_simulationFlags` アノテーションがモデル内に含まれていれば、それらの設定はシミュレーション時に反映されます。
- シミュレーション設定ウィンドウを介して行われたすべての変更は、セッション全体にわたって保持されます。 今後のセッションで同じ設定を使用する場合は、それらを `experiment` と `__OpenModelica_simulationFlags` 内に保存する必要があります。

OMEdit シミュレーションセットアップは、次の方法で起動できます。

- メニューから [シミュレーション]> [シミュレーション設定] を選択します。 （ModelWidget でアクティブになっているモデルが必要です）
- シミュレーション設定ツールバーボタンをクリックします。 （ModelWidget でアクティブになっているモデルが必要です）
- ライブラリブラウザからモデルを右クリックし、[Simulation Setup(シミュレーションのセットアップ)] を選択します。

2.13.1 General(全般) タブ

- Simulation Interval(解析間隔)
- *Start Time(**開始時間**)* - シミュレーション開始時刻。
- *Stop Time(**停止時間**)* - シミュレーション停止時間。
- *Number of Intervals(**計算回数**)* - シミュレーションプロット数。
- *Interval(**間隔**)* - 1 回のタイムステップ間の長さ（すなわち、ステップサイズ）
- インタラクティブシミュレーション
- ステップを使ったシミュレーション（インタラクティブなシミュレーションを同期させます。パフォーマンスを犠牲にして、より良い曲線をプロットします）
- シミュレーションサーバポート
- Integration(積分)
  - *Method(**手法**)* - シミュレーションソルバー。ソルバーの詳細については、*Integration Methods* セクションを参照してください。
  - *Tolerance(**許容値**)* - シミュレーションの許容誤差。
  - *Jacobian(**ヤコビアン**)* - 使用するヤコビアン手法。
  - DASSL/IDA Options(DASSL/IDA オプション)
  - *Root Finding(**ルートの検索**)* - dassl の内部ルート探索手法のアクティブ化。
  - *Restart After Event(**イベントの後でリスタート**)* - イベント発生後に dassl の再実行。
  - *Initial Step Size(**初期ステップサイズ**)*
  - *Maximum Step Size(**最大ステップサイズ**)*
  - *Maximum Integration Order(**最大積分次数**)*
- *C/C++ Compiler Flags (Optional)* - C/C++コンパイラのオプション
- *Number of Processors(**プロセッサ数**)* - シミュレーションをビルドするために使用されるプロセッサの数。
- *Build Only(**構築のみ**)* - ビルドのみ実行。
- *Launch Transformational Debugger(**変換デバガを起動する**)* - 変換デバッガを起動。
- *Launch Algorithmic Debugger(**アルゴリズムデバガを起動する**)* - アルゴリズムデバッガを起動。
- *Launch Animation* - 3D アニメーションウィンドウを起動。

2.13.2 Output(出力) タブ

- *Output Format(**出力書式**)* - シミュレーション結果ファイルの出力形式。
- *Single Precision* - 単精度で出力結果 (mat 形式のみ有効)。
- *File Name Prefix (Optional)(**ファイル名のプリフィックス（オプション）**)* - 入力された文字列は出力ファイル名の接頭辞として使用。
- *Result File (Optional)(**結果ファイル（オプション）**)* - シミュレーション結果のファイル名。
- *Variable Filter (Optional)(**変数フィルター（オプション）**)*
- *Protected Variables(**保護された変数**)* - 結果ファイルにプロテクトされた変数を追加します。
- *Equidistant Time Grid(**等間隔の時間グリッド**)* - dassl で指定された内部ステップを出力する代わりに、ステップサイズまたは計算回数で指定された等間隔時間グリッドとして結果を補間。
- *Store Variables at Events(**イベントでの編集を保存**)* - イベントが発生したタイミング毎で変数値を結果ファイルに追加
- *Show Generated File(**生成ファイルを表示**)* - ダイアログボックスに生成されたファイルを表示。

2.13.3 Simulation Flags(シミュレーションフラグ) タブ

- *Model Setup File (Optional)(**モデル設定ファイル（オプション）**)* - 生成されたシミュレーションコードに指定した XML ファイルを適用。
- *Initialization Method (Optional)(**初期化方法（オプション）**)* - 初期化方法を指定。
- *Equation System Initialization File (Optional)(**連立方程式初期化ファイル（オプション）**)* - モデルの初期化のための外部ファイルを指定。
- *Equation System Initialization Time (Optional)(**連立方程式初期化時刻（オプション）**)* - モデルの初期化のための時刻を指定。
- *Clock (Optional)(**クロック（オプション）**)* - 使用するクロックの種類を指定。
- *Linear Solver (Optional)(**線形ソルバー（オプション）**)* - 線形ソルバーを指定。
- *Non Linear Solver (Optional)(**非線形ソルバー（オプション）**)* - 非線形ソルバーを指定。
- *Linearization Time (Optional)(**初期化時刻（オプション）**)* - モデルの線形化を実行する時間を指定。
- *Output Variables (Optional)(**出力変数（オプション）**)* - シミュレーションの終了時に指定した変数を出力。
- *Profiling(**プロファイリング**)* - プロファイリングした HTML ファイルを作成。
- *CPU Time* - 結果ファイルに CPU 時間をダンプ。
- 出力はすべての警告 - **すべての警告を有効にします**。
- *Logging (Optional)(**ログ記録（オプション）**)*
- *stdout* - 標準出力ストリーム。 このストリームは常にアクティブであり、-lv=-stdout で無効にできます

- *assert* - このストリームは、常にアクティブで-lv=-assert で無効にすることができます
- *LOG_DASSL* - dassl ソルバーについての追加情報。
- *LOG_DASSL_STATES* - すべての dassl の呼び出しで状態を出力。
- *LOG_DEBUG* - 追加のデバッグ情報。
- *LOG_DSS* - dynamic state selection についての情報を出力。
- *LOG_DSS_JAC* - dynamic state のヤコビアンを出力。
- *LOG_DT* - ダイナミックティアリングに関する追加情報。
- *LOG_DT_CONS* - ダイナミックティアリング（ローカルおよびグローバルな制約）についての追加情報。
- *LOG_EVENTS* - イベントイタレーション中の追加情報。
- *LOG_EVENTS_V* - イベントシステムの冗長なログ。
- *LOG_INIT* - 初期化中の追加情報。
- *LOG_IPOPT* - Ipopt からの情報。
- *LOG_IPOPT_FULL* - Ipopt からより多くの情報を出力。
- *LOG_IPOPT_JAC* - Ipopt でヤコビ行列をチェック。
- *LOG_IPOPT_HESSE* - Ipopt とヘッセ行列をチェック。
- *LOG_IPOPT_ERROR* - 最適化中で最大のエラーを出力。
- *LOG_JAC* - dassl によって使用されるヤコビ行列を出力。
- *LOG_LS* - 線形システムのログ。
- *LOG_LS_V* - 線形システムの冗長なログ。
- *LOG_NLS* - 非線形システムのログ。
- *LOG_NLS_V* - 非線形システムの冗長なログ。
- *LOG_NLS_HOMOTOPY* - 非線形システムのためのホモトピーソルバーのログ。
- *LOG_NLS_JAC* - 非線形システムのヤコビアンを出力。
- *LOG_NLS_JAC_TEST* - 非線形システムの解析的ヤコビアンのテスト。
- *LOG_NLS_RES* - 残差関数のすべての評価を出力。
- *LOG_NLS_EXTRAPOLATE* - 外挿プロセスに関するデバッグ情報を出力。
- *LOG_RES_INIT* - 初期の残差を出力。
- *LOG_RT* - リアルタイムプロセスに関する追加情報。
- *LOG_SIMULATION* - シミュレーション・プロセスに関する追加情報。
- *LOG_SOLVER* - ソルバー・プロセスに関する追加情報。
- *LOG_SOLVER_V* - 統合プロセスに関する詳細情報。

- *LOG_SOLVER_CONTEXT* - ソルバ処理中のコンテキスト情報。
- *LOG_SOTI* - 初期化の最終解析結果。
- *LOG_STATS* - タイマー/イベント/ソルバーについての追加の統計。
- *LOG_STATS_V* - LOG_STATS のための追加の統計情報。
- *LOG_SUCCESS* - このストリームは常にアクティブで、=-lv=-LOG_SUCCESS で無効にすることができます。
- *LOG_UTIL*.
- *LOG_ZEROCROSSINGS* - ゼロクロッシングに関する追加情報。
- *Additional Simulation Flags (Optional)*(**追加シミュレーションフラグ（オプション）**) - 任意の他のシミュレーションフラグを指定します。

2.13.4 アーカイブされたシミュレーションタブ

すでに終了または実行中のシミュレーションのリストを表示します。 それらのいずれかをダブルクリックすると、シミュレーション出力ウィンドウが開きます。

2.14 シミュレーション結果をプロット

モデルのシミュレーションが成功すると、プロットの候補となるインスタンス変数を含む結果ファイルが生成されます。 変数ブラウザには、そのようなインスタンス変数のリストが表示されます。 各変数にはチェックボックスがあり、チェックすると変数がプロットされます。 Figure 2.8 を参照してください。

2.14.1 プロットの種類

プロットタイプは、アクティブなプロットウィンドウに依存します。デフォルトでは、プロットタイプは、時間プロットです。

時間プロット

シミュレーション時間を横軸にして変数をプロットします。 [新規プロットウィンドウ] ツールバーボタン () をクリックすると、複数のタイムプロットウィンドウを作成できます。

プロットパラメトリック

x の関数として *y* を使用して、横軸を変数 x と縦軸を変数 y として 2 次元パラメトリック図を描画します。[新規 X-Y(パラメトリック) プロットウィンドウ] ツールバーボタン () をクリックすると、複数のプロットパラメトリックウィンドウを作成できます。

配列のプロット

配列要素のインデックスを横軸として、対応する要素の値を縦軸に配列変数をプロットします。 時間は、変数ツリーの上にあるスライダーによって変更できます。 配列がモデルに存在する場合、変数ツリーに主要な配列ノードがあります。 この配列を配列プロットとしてプロットするには、主ノードを一致させます。主ノードは、特定の配列要素に展開できます。 タイムプロットで単一の要素をプロットするには、要素を一致させます。 New Array Plot Window ツールバーボタン () を使用して、新しい ArrayPlot ウィンドウが開きます。

アレイパラメトリックプロット

x 軸に最初の配列要素の値をプロットし、y 軸に 2 番目の配列要素の値をプロットします。 時間は、変数ツリーの上にあるスライダーによって変更できます。 新しい配列パラメトリックプロットを作成するには、[New Array Parametric Plot Window] ツールバーボタン () を押してから、x 軸にプロットする変数ツリービューの主要な配列ノードを適用し、 次に y 軸にプロットされる主要な配列ノードを適用します。

ダイアグラムウィンドウ

アクティブな ModelWidget を読み取り専用の図として表示します。 ダイアグラムウィンドウは 1 つしか表示できません。 それを表示するには、ダイアグラムウィンドウのツールバーボタン () をクリックします。

2.15 再シミュレートするモデル

Variables Browser(変数ブラウザ) を使用すると、変更可能なパラメータを変更して再シミュレーションできます。 パラメータ値を変更した後、ユーザーはツールバーの再シミュレーションボタン () をクリックするか、変数ブラウザでモデルを右クリックしてメニューから再シミュレーションを選択できます。

2.16 3D オブジェクトの可視化

OpenModelica 1.11 以降、OMEdit には 3D 可視化が組み込まれています。これは、3D 可視化用のサードパーティライブラリ（Modelica3D のような）に代わるものです。

2.16.1 可視化の実行

3D 可視化機能は OpenSceneGraph に基づいています。 可視化を実行するには、ライブラリブラウザでクラスを右クリックし、 Figure 2.11 に示すように *Simulate with Animation* をクリックします。

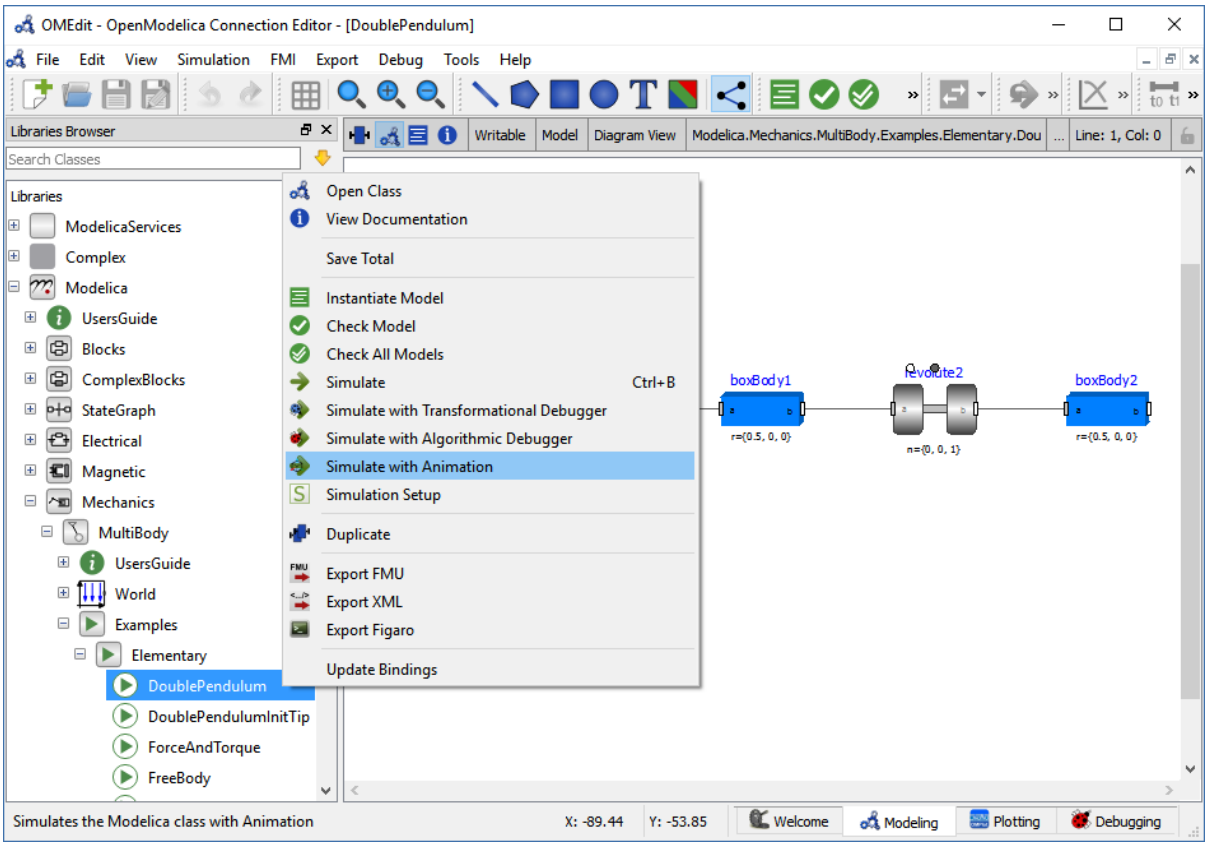


Figure 2.11: アニメーション機能を用いた OMEdit のシミュレーション

メニューから [シミュレーション]> [Simulate with Animation] を使用して可視化を実行することもできます。

アニメーションモードでモデルをシミュレートする場合、フラグ *+d=visxml* が設定されます。したがって、コンパイラは、マルチボディシェイプに関するすべての情報が記述されたファイル *_visual.xml* (シーン説明ファイル) を生成します。このファイルは、マルチボディシステムのアニメーションに必要なすべての変数を参照しています。 *+d=visxml* でシミュレーションする場合、コンパイラーは常にこれらの変数の結果を生成します。

2.16.2 可視化の表示

モデルのシミュレーションが成功すると、 Figure 2.12 に示すように可視化ウィンドウが自動的に表示されます。

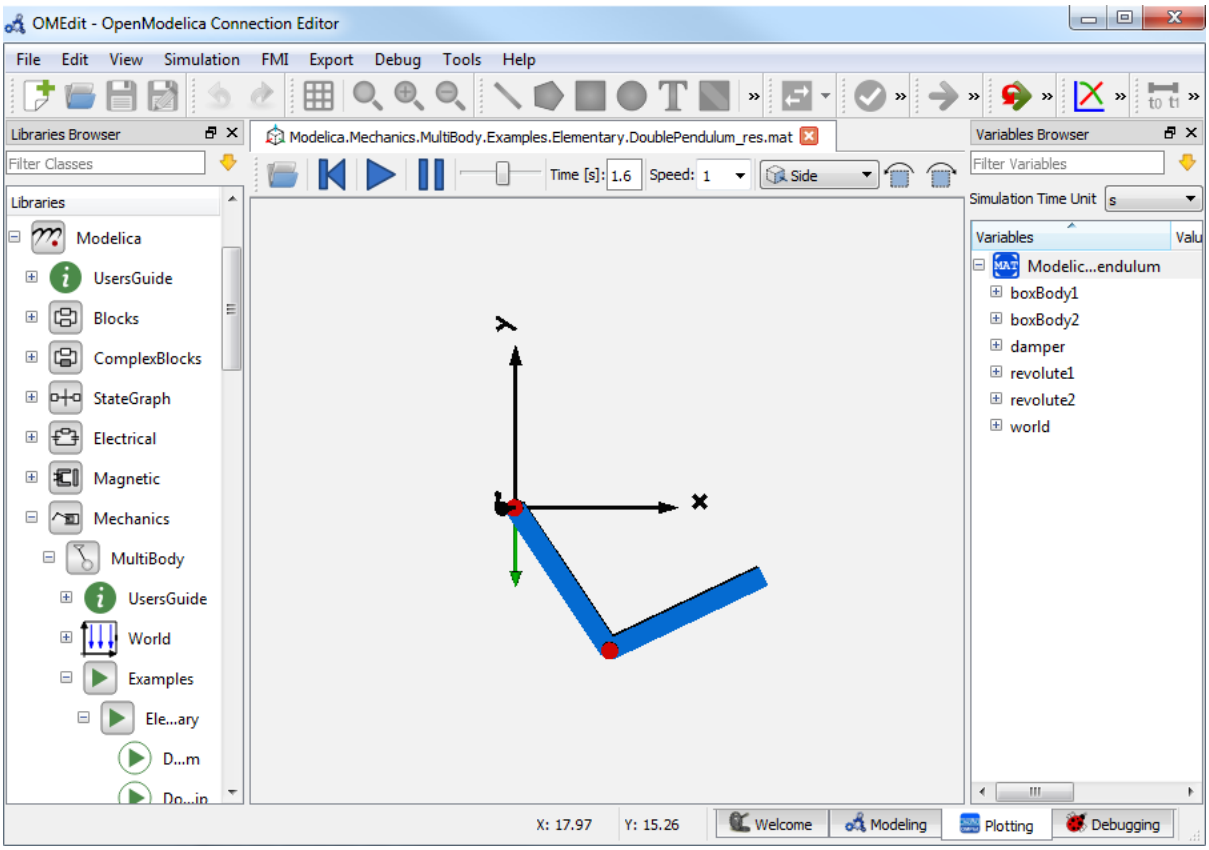


Figure 2.12: OMEdit の 3D 可視化

アニメーションは、 *Play* ボタンを押すと起動します。 アニメーションは、stopTime まで、または *Pause* ボタンが押されるまで再生されます。 *previous* (訳注：OMEdit を確認すると *Initialize*) ボタンを押すと、アニメーションは最初の時点にジャンプします。 **タイムスライダー** を動かすか、**タイムボックス** にシミュレーション時間を挿入することで、ポイントを選択できます。リアルタイムに関連するアニメーションの速度の倍数は、**速度ダイアログ** で設定できます。他のアニメーションは、[open file] ボタンを使用し、対応するシーン記述ファイルを含む結果ファイルを選択することで開くことができます。

次のように 3D カメラビューを操作することができます。

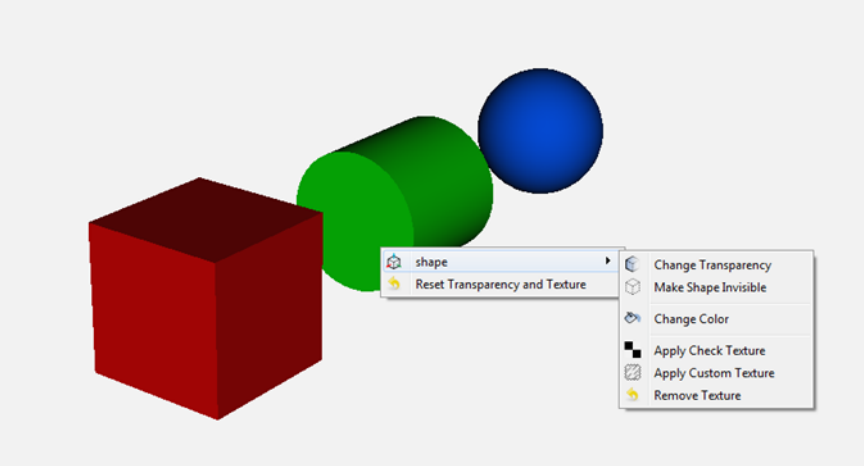
操作	キー	マウスアクション
拡大/縮小する	無し	ホイール
拡大/縮小する	マウスの右ボタンのホールド	上下
上/下/左/右に移動します	マウスの中ボタンのホールド	マウスを動かします
上/下/左/右に移動します	マウスの左右ボタンのホールド	マウスを動かします
回転	マウスの左ボタンのホールド	マウスを動かします
シェイプコンテキストメニュー	マウスの右ボタン+ Shift キー	

定義済みのビュー（Isometric(等角投影)、Side、Front、Top）を選択することができ、回転ボタンを使用し

て表示シーンを時計または反時計回りに 90° 傾けることができます。

2.16.3 追加の可視化機能

ビューアに表示される形状は、Shift +右クリックで選択できます。 シェイプを選択すると、追加の視覚化機能を提供するコンテキストメニューがポップアップ表示されます。

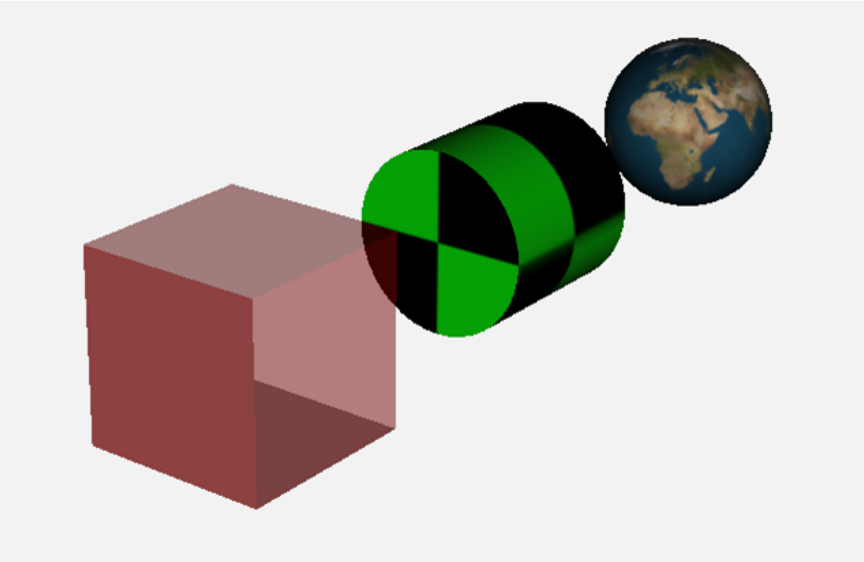


次の機能を選択することができます。

メニュー	説明
Change Transparency	形状は、透明または不透明のいずれかになります。
Make Shape Invisible	形状が見えなくなります。
Change Color	カラーダイアログがポップアップして形状の色を設定することができます。
Apply Check Texture	チェックしたテクスチャは、形状に適用されます。
Apply Custom Texture	ファイル選択ダイアログがポップアップし、画像ファイルをテクスチャとして選択することができます。
Remove Texture	形状の現在のテクスチャを削除します。

2.17 リアルタイム FMU のアニメーション

結果ファイルの代わりに、OMEdit は Functional Mock-up Units をロードして、マルチボディシステムのアニメーションのデータを取得できます。 アニメーションプロットビューから mat ファイルを開くのと同じように、FMU ファイルを開くことができます。 必然的に、FMU は *+d=visxml* フラグをアクティブにして生成する必要があります。これにより、シーン記述ファイルが FMU と同じディレクトリに生成されます。現在、FMU1.0 および FMU2.0 モデルエクスチェンジのみがサポートされています。 FMU を選択すると、シミュレーション設定ウィンドウがポップアップして、ソルバーとステップサイズを選択します。 その後、モデルが初期化され、再生ボタンを押すことでシミュレーションできます。



2.17.1 FMU のインタラクティブリアルタイムアニメーション

FMU は、リアルタイムのユーザー操作でシミュレートできます。 考えられる解決策は、Modelica_DeviceDrivers ライブラリ（https://github.com/modelica/Modelica_DeviceDrivers）のインタラクションモデルをモデルに装備することです。 リアルタイム同期は OMEdit によって行われるため、追加の時間同期モデルは必要ありません。

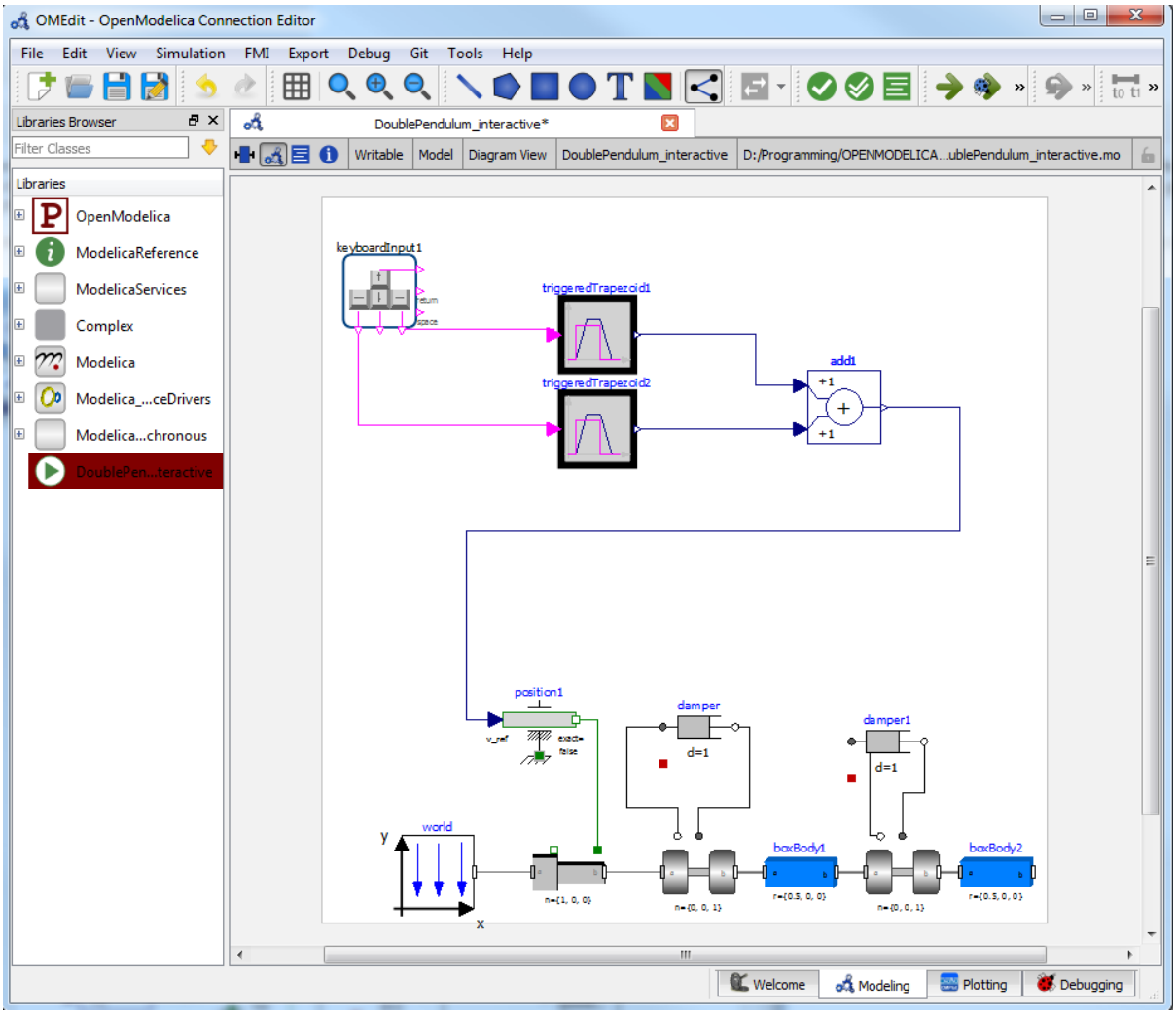
2.18 インタラクティブシミュレーション

> **警告:** インタラクティブなシミュレーションは、実験的な機能です。

インタラクティブシミュレーションは、シミュレーション設定の *General* タブでインタラクティブシミュレーションを選択することで有効になります。

実行には、非同期と同期（ステップでシミュレート）の 2 つの主要なモードがあります。 違いは、同期（ステップモード）では、OMEdit は、シミュレーションが実行する必要のある各ステップのコマンドをシミュレーションに送信することです。 非同期モードは、シミュレーションに変数値をリアルタイムで実行およびサンプリングするように指示するだけです。 シミュレーションが非常に高速に実行される場合、サンプリングされる値は少なくなります。

非同期モードで実行する場合、モデルをリアルタイムでシミュレートすることができます（シミュレーションフラグ *-rt* と同様にスケーリング係数を用いますが、 インタラクティブシミュレーションの間スケーリング係数を変更する機能を使用します）。同期モードでは、シミュレーションの速度はリアルタイムに直接対応しません。


OMEdit - OpenModelica Connection Editor
File Edit View Simulation FMI Export Debug Git Tools Help
Libraries Browser
Filter Classes
Libraries
OpenModelica
ModelicaReference
ModelicaServices
Complex
Modelica
Modelica_...ceDrivers
Modelica...chronous
DoublePen...teractive
DoublePendulum_interactive*
Writable
Model
Diagram View
DoublePendulum_interactive
D:/Programming/OPENMODELICA...ublePendulum_interactive.mo
keyboardInput1
triggeredTrapezoid1
triggeredTrapezoid2
add1
+1
+1
position1
damper
d=1
damper1
d=1
world
boxBody1
boxBody2
Welcome
Modeling
Plotting
Debugging

2.19 ユーザー定義シェイプの作成方法 - アイコン

ユーザーは OMEdit で利用可能な形状作成ツールを使用して、独自の図形を作成することができます。

- *Line Tool*(**線**) - 線を描画します。 線は最低 2 点で作成されます。 線を作成するには、ユーザーは最初にツールバーから線ツールを選択し、次にアイコン/ダイアグラムビューをクリックします。 これにより、線の作成が開始されます。 ユーザーがアイコン/ダイアグラムビューをもう一度クリックすると、新しいラインポイントが作成されます。 線の作成を完了するには、ユーザーはアイコン/ダイアグラムビューをダブルクリックする必要があります。
- *Polygon Tool*(**多角形**) - ポリゴンを描画します。 ポリゴンは、ラインが作成されるのと同様の方法で作成されます。 線と多角形の唯一の違いは、多角形に 2 つの点が含まれている場合は線のように見え、多角形に 3 つ以上の点が含まれている場合は閉じた多角形になることです。
- *Rectangle Tool*(**長方形**) - 長方形を描画します。 長方形には 2 つのポイントのみが含まれ、最初のポイントは開始ポイントを示し、2 番目のポイントは終了ポイントを示します。 長方形を作成するには、ユーザーはツールバーから長方形ツールを選択し、アイコン/ダイアグラムビューをクリックする必要があります。このクリックが長方形の最初のポイントになります。 長方形の作成を終了するには、ユーザーは長方形を完成させたいアイコン/ダイアグラムビューをもう一度クリックする必要があります。 2 回目のクリックは、長方形の 2 番目の点になります。
- *Ellipse Tool*(**楕円**) - 楕円を描画します。 楕円は、長方形が作成されるのと同様の方法で作成されます。
- *Text Tool* - テキストラベルを描画します。
- *Bitmap Tool*(**ビットマップ**) - ビットマップ画像を描画します。

シェイプツールはツールバーの中に配置されています。 Figure 2.13 を参照してください。

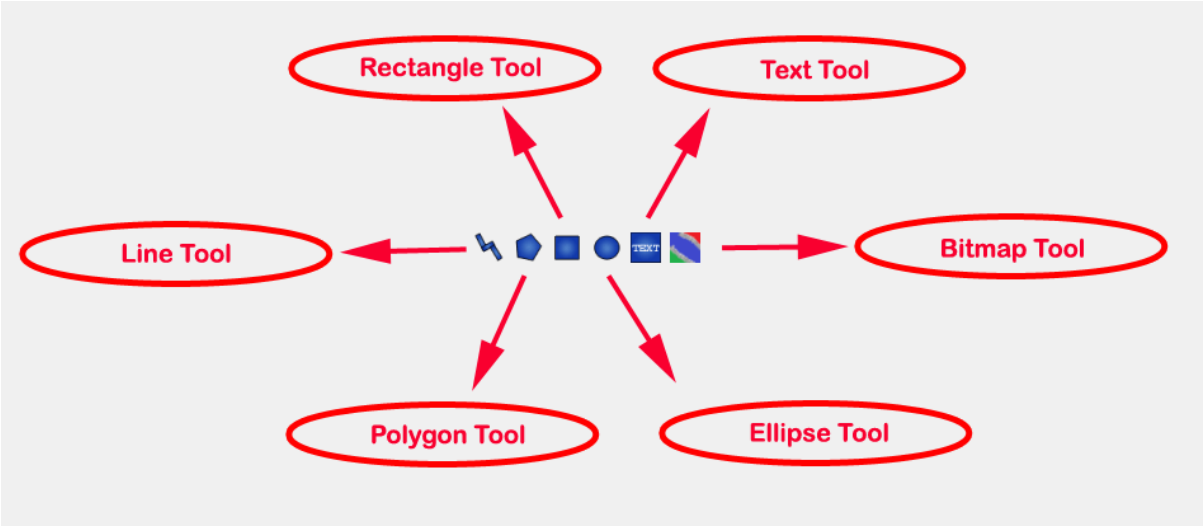


Figure 2.13: ユーザー定義形状

ユーザーは、任意の形状ツールを選択して、アイコン/ダイアグラムビューで描画を開始できます。 モデルウィジェットのダイアグラムビューで作成されたシェイプはダイアグラムの一部であり、アイコンビューで作成されたシェイプはモデルのアイコンの表示になります。

たとえば、ユーザーが testModel という名前のモデルを作成し、長方形ツールを使用して長方形を追加し、ポリゴンツールを使用してポリゴンをモデルのアイコンビューに追加したとします。 モデルの Modelica テ

キストは次のように表示されます。

```
model testModel
  annotation(Icon(graphics = {Rectangle(rotation = 0, lineColor = {0,0,255},␣
↪fillColor = {0,0,255}, pattern = LinePattern.Solid, fillPattern = FillPattern.
↪None, lineThickness = 0.25, extent = {{ -64.5,88},{63, -22.5}}),Polygon(points =
↪{{ -47.5, -29.5},{52.5, -29.5},{4.5, -86},{ -47.5, -29.5}}, rotation = 0,␣
↪lineColor = {0,0,255}, fillColor = {0,0,255}, pattern = LinePattern.Solid,␣
↪fillPattern = FillPattern.None, lineThickness = 0.25)}));
end testModel;
```

上記の testModel のコードスニペットでは、長方形とポリゴンがモデルのアイコンアノテーションに追加されています。 同様に、モデルのダイアグラムビューに描画されたユーザー定義の形状は、モデルのダイアグラムアノテーションに追加されます。

2.20 ドキュメントのグローバルヘッドセクション

パッケージ内に含まれるクラスに同じスタイルまたは同じ JavaScript を使用する場合は、パッケージの `Documentation` アノテーション内に `__OpenModelica_infoHeader` アノテーションを定義します。例えば、

```
package P
  model M
    annotation(Documentation(info="<html>
      <a href=\"javascript:HelloWorld()\">Click here</a>
    </html>"));
  end M;
 annotation(Documentation(__OpenModelica_infoHeader="
     <script type=\"text/javascript\">
       function HelloWorld() {
         alert(\"Hello World!\");
       }
     </script>"));
end P;
```

上記の例では、モデル `M` は javascript 関数 `HelloWorld` を定義する必要はありません。`__OpenModelica_infoHeader` を使用してパッケージレベルで一度だけ定義すればよく、パッケージに含まれるすべてのクラスがそれを使用できます。

さらに、Modelica URI を使用して、ファイルの場所からスタイルと JavaScript を追加できます。 例：

```
package P
  model M
    annotation(Documentation(info="<html>
      <a href=\"javascript:HelloWorld()\">Click here</a>
    </html>"));
  end M;
 annotation(Documentation(__OpenModelica_infoHeader="
     <script type=\"text/javascript\">
        src=\"modelica://P/Resources/hello.js\">
       }
     </script>"));
end P;
```

ファイル `Resources/hello.js` を含ませる場合

```
function HelloWorld() {
  alert("Hello World!");
}
```

2.21 オプション

OMEdit を使用すると、ユーザーは OMEdit のさまざまなセッションで記憶されるいくつかのオプションを保存できます。 オプションダイアログは、オプションの読み取りと書き込みに使用できます。

2.21.1 General(全般)

- General(全般)
- *Language(*言語*)* - アプリケーションの言語を設定します。
- *Working Directory(*作業ディレクトリ*)* - アプリケーションの作業ディレクトリを設定します。すべてのファイルはこのディレクトリに生成されます。
- *Toolbar Icon Size(*ツールバーのアイコンサイズ*)* - ツールバーのアイコンのサイズを設定します。
- *Preserve User's GUI Customizations(*ユーザ *GUI* カスタマイズの保存*)* -true の場合、OMEdit はウィンドウとツールバーの位置とサイズを記憶します。
- *Terminal Command(*ターミナルコマンド*)* - ターミナルコマンドを設定します。 ユーザーが [ツール]> [ターミナルを開く] をクリックすると、このコマンドが実行されます。
- *Terminal Command Arguments(*ターミナルコマンドの引数*)* - ターミナルコマンド引数を設定します。
- *Hide Variables Browser* - プロットプロスペクティブへ切り替えるときに変数ブラウザを非表示にします。
- *Activate Access Annotations* - 暗号化されていないライブラリのアクセスアノテーションをアクティブにします。 アクセスアノテーションは、暗号化されたライブラリに対して常にアクティブです。
- Libraries Browser(ライブラリブラウザ)
- *Library Icon Size(*ライブラリアイコンサイズ*)* - ライブラリアイコンのサイズを設定します。
- *Show Protected Classes*（プロテクトされているクラスを表示） - 有効にすると、ライブラリブラウザに保護されたクラスも一覧表示されます。
- モデルビューモード
- *Tabbed View/SubWindow View(*タブ形式ビュー/サブウィンドウビュー*)* - モデリングのためのビューモードを設定します。
- Default View(デフォルトビュー)
- *Icon View(*アイコンビュー*)/DiagramView(*ダイアグラムビュー*)/Modelica Text View(*テキストビュー*)/Documentation View(*ドキュメントビュー*)* -preferredView アノテーションが定義されていない場合、ユーザーがライブラリブラウザーでクラスをダブルクリックしたときにここで設定されたビューが表示されます。

- *Enable Auto Save(**自動保存を使用**)* - 自動保存を有効にします。
- *Auto Save interval*（**自動保存間隔**） - 自動保存の時間間隔を設定します。設定可能な最小間隔値は 60 秒です。
- *Enable Auto Save for single classes* - 1 つのファイルに保存された 1 つのクラスの自動保存を有効にします。
- *Enable Auto Save for one file packages* - 1 つのファイルに保存されたパッケージの自動保存を有効にします。
- Welcome Page（起動ページ）
- *Horizontal View(**水平ビュー**)/Vertical View(**垂直ビュー**)* - 起動ページのビューモードを設定します。
- *Show Latest News(**最新情報を表示**)* - true の場合は、最新のニュースを表示します。

2.21.2 Libraries(ライブラリ)

- *System Libraries*（**システムライブラリ**） - OMEdit が起動するたびにロードされるシステムライブラリのリスト。
- *Force loading of Modelica Standard Library(Modelica Standard Library* **の強制ロード**) - true の場合、Modelica と ModelicaReference は、ユーザーがシステムライブラリのリストから削除した場合でも常にロードされます。
- *Load OpenModelica library on startup* - true の場合、OMEdit 起動時に OpenModelica パッケージがロードされます。
- *User Libraries(**ユーザライブラリ**)* - OMEdit が起動時にロードされるユーザーライブラリ/ファイルのリスト。

2.21.3 Text Editor

- Format(フォーマット)
- *Line Ending* - ファイルの改行コードを設定します。
- *Byte Order Mark (BOM)* - ファイルの BOM(バイト順マーク) を設定します。
- Tabs and Indentation(タブとインデント)
- *Tab Policy(**タブのポリシー**)* - タブポリシーをスペースまたはタブのみに設定します。
- *Tab Size(**タブのサイズ**)* - タブのサイズを設定します。
- *Indent Size(**インデントサイズ**)* - インデントサイズを設定します。
- Syntax Highlight and Text Wrapping(構文のハイライトとテキストのラッピング)
    - *Enable Syntax Highlighting(**構文ハイライトを使用**)* - 構文ハイライトの有効/無効。

  - *Enable Code Folding* - コードの折り畳みを有効/無効にします。 コードの折り畳みを有効にすると、複数行のアノテーションがコンパクトなアイコン ("..." を含む長方形) に折りたたまれます。行が折りたたまれると「 "+" 記号」を含むマーカーが行の左側で使用できるようになりコードを自由に展開/折りたたむことができます。
  - *Match Parentheses within Comments and Quotes(**コメントと引用でのカッコの整合**)* - コメントと引用符内の括弧の一致を有効/無効にします。
  - *Enable Line Wrapping(**行の折り返しを使用**)* - 行の折り返しを有効/無効にします。
- Autocomplete
- *Enable Autocomplete* - オートコンプリートを有効/無効にします。
- Font
- *Font Family(**フォントの種類**)* - 利用可能なフォント名の一覧を表示します。テキストエディタのフォントを設定します。
- *Font Size(**フォントサイズ**)* - テキストエディタのフォントサイズを設定します。

2.21.4 Modelica Editor

- *Preserve Text Indentation(**テキストインデントを保持**)* -true の場合は *diffModelicaFileListings* API の呼び出しを使用し、それ以外の場合は OMC プリティプリントを使用します。
- Colors
- *Items(**アイテム**)* - 構文ハイライトコードのカテゴリリスト。
- *Item Color(**アイテムの色**)* - 選択した項目の色を設定します。
- *Preview(**プレビュー**)* - 構文ハイライトの表示デモ。

2.21.5 MetaModelica Editor

- Colors
- *Items(**アイテム**)* - 構文ハイライトコードのカテゴリリスト。
- *Item Color(**アイテムの色**)* - 選択した項目の色を設定します。
- *Preview(**プレビュー**)* - 構文ハイライトの表示デモ。

2.21.6 CompositeModel Editor

- Colors
- *Items(**アイテム**)* - 構文ハイライトコードのカテゴリリスト。
- *Item Color(**アイテムの色**)* - 選択した項目の色を設定します。
- *Preview(**プレビュー**)* - 構文ハイライトの表示デモ。

2.21.7 C/C++ Editor

- Colors
- *Items(**アイテム**)* - 構文ハイライトコードのカテゴリリスト。
- *Item Color(**アイテムの色**)* - 選択した項目の色を設定します。
- *Preview(**プレビュー**)* - 構文ハイライトの表示デモ。

2.21.8 Graphical Views(グラフィックビュー)

- Extent(範囲)
- *Left(**左**)* - ビューの左側の範囲ポイントを定義します。
- *Bottom(**下**)* - ビューの下側の範囲ポイントを定義します。
- *Right(**右**)* - ビューの右側の範囲ポイントを定義します。
- *Top(**左**)* - ビューの上側の範囲ポイントを定義します。
- Grid(グリッド)
- *Horizontal(**水平**)* - ビューグリッドの水平方向のグリッドサイズを定義します。
- *Vertical(**垂直**)* - ビューグリッドの垂直方向のグリッドサイズを定義します。
- Component(コンポーネント)
- *Scale factor(**倍率**)* - ビューにドラッグされたコンポーネントに対するアイコンの初期倍率を定義します。
- *Preserve aspect ratio(**アスペクト比を保持**)* - true の場合、アイコンのサイズを変更したときのアスペクト比が保持されます。

2.21.9 シミュレーション

- シミュレーション
  - Translation Flags
  - *Matching Algorithm(***マッチングアルゴリズム***)* - シミュレーションのためのマッチングアルゴリズムを設定します。
  - *Index Reduction Method(***インデックス低減手法***)* - シミュレーションのためのインデックスの低減方法を設定します。
  - *Show additional information from the initialization process* - 初期化プロセスから得られる情報を出力します。
  - *Evaluate all parameters (faster simulation, cannot change them at runtime)* - シミュレーションがより効率的になりますが、再計算 (Re-Simulation) に代わって、パラメータを変更する場合はモデルを再コンパイルする必要があります。
  - *Enable analytical jacobian for non-linear strong components* - ユーザー定義の関数の呼び出しなしで、非線形の強いコンポーネントに対する解析的なヤコビアンを有効にします。
  - *Enable pedantic debug-mode, to get much more feedback*
  - *Enable parallelization of independent systems of equations (Experimental)*
  - *Enable experimental new instantiation phase*
  - *Additional Translation Flags* - 変換フラグを設定します。 *Options* を参照してください。
  - *Target Language(***ターゲット言語***)* - Modelica 言語を基に生成されるプログラミング言語を設定します。
  - *Target Build* - 生成されたコードのコンパイルに使用されるコンパイラを設定します。
  - *C Compiler* - 生成されたコードをコンパイルするための C コンパイラを設定します。
  - *CXX Compiler* - 生成されたコードをコンパイルするための CXX コンパイラを設定します。
  - *Ignore __OpenModelica_commandLineOptions annotation* - true の場合、シミュレーションの実行中に__OpenModelica_commandLineOptions アノテーションを無視します。
  - *Ignore __OpenModelica_simulationFlags annotation* - true の場合、シミュレーションの実行中に__OpenModelica_simulationFlags アノテーションを無視します。
  - *Save class before simulation(***シミュレーション前にクラスを保存***)* - true の場合は、常にシミュレーションを実行する前にクラスを保存します。
  - *Switch to plotting perspective after simulation* - true の場合、GUI はシミュレーション後に常にプロットパースペクティブに切り替わります。
  - *Close completed simulation output windows before simulation* - true の場合、新しいシミュレーションを開始する前に、完了したシミュレーション出力ウィンドウを閉じます。
  - *Delete intermediate compilation files* - true の場合、コンパイル中に生成されたファイルは自動的に削除されます。

- *Delete entire simulation directory of the model when OMEdit is closed* - true の場合、シミュレーションディレクトリ全体が終了時に削除されます。
- Output(出力)
- *Structured(**構造化**)* - ツリー構造の形でシミュレーション出力を表示します。
- *Formatted Text(**書式ありテキスト**)* - シミュレーション出力をフォーマットされたテキストの形式で表示します。

2.21.10 Messages(メッセージ)

- General(全般)
- *Output Size(**出力サイズ**)* - メッセージブラウザが持つことができる最大行数を指定します。 最大行数以上の場合、メッセージは最初から削除されます。
- *Reset messages number before simulation* - シミュレーションを開始する前にメッセージカウンターをリセットします。
- Font and Colors
- *Font Family(**フォント種類**)* - メッセージのフォントを設定します。
- *Font Size(**フォントサイズ**)* - メッセージのフォントサイズを設定します。
- *Notification Color(**通知用色**)* - 通知メッセージのテキストの色を設定します。
- *Warning Color(**警告色**)* - 警告メッセージのテキストの色を設定します。
- *Error Color(**エラー色**)* - エラーメッセージのテキストの色を設定します。

2.21.11 Notifications(通知)

- Notifications(通知)
- *Always quit without prompt(**プロンプトなしで常に終了**)* -true の場合、OMEdit はユーザーにプロンプトを表示せずに終了します。
- *Show item dropped on itself message(**自分自身のアイテムがドロップされた場合のメッセージ表示**)* - true の場合、クラスが自分自身にドラッグアンドドロップされたときにメッセージがポップアップ表示されます。
- *Show model is defined as partial and component will be added as replaceable message(**モデルが** partial **で** replaceable(**置換可能**) **として追加された場合のメッセージ表示**)* - true の場合、パーシャルクラスが別のクラスに追加されるとメッセージがポップアップします。
- *Show component is declared as inner message(**コンポーネントが** inner **として宣言された場合のメッセージ表示**)* - true の場合、inner コンポーネントが別のクラスに追加されるとメッセージがポップアップ表示されます。

- *Show save model for bitmap insertion message(**ビットマップが挿入されたモデル保存時のメッセージ表示**)* - true の場合、ユーザーが保存されていないクラスにローカルディレクトリからビットマップを挿入しようとすると、メッセージがポップアップ表示されます。
- *Always ask for the dragged component name* - true の場合、ユーザーがコンポーネントをグラフィカルビューにドラッグアンドドロップすると、メッセージがポップアップ表示されます。
- *Always ask for what to do with the text editor error* - true の場合、テキストエディタにエラーがあるとメッセージが常にポップアップ表示されます。

2.21.12 Line Style(線のスタイル)

- Line Style(線のスタイル)
- *Color(**色**)* - 線の色を設定します。
- *Pattern(**パターン**)* - 線のパターンを設定します。
- *Thickness(**太さ**)* - 線の太さを設定します。
- *Start Arrow(**始点側矢印**)* - 始点側矢印の形状を設定します。
- *End Arrow(**終点側矢印**)* - 終点側矢印の形状を設定します。
- *Arrow Size(**矢印サイズ**)* - 始点と終点の矢印のサイズを設定します。
- *Smooth(**なめらかさ**)* - true の場合、線はベジェ曲線として描画されます。

2.21.13 Fill Style(塗りつぶしタイプ)

- Fill Style(塗りつぶしタイプ)
- *Color(**色**)* - 塗りつぶしの色を設定します。
- *Pattern(**パターン**)* - 塗りつぶしのパターンを設定します。

2.21.14 Plotting(プロット)

- General(全般)
- *Auto Scale(**自動スケール**)* - プロットを自動スケールするかどうかを設定します。
- Plotting View Mode
- *Tabbed View/SubWindow View(**タブ形式/サブウィンドウビュー**)* - プロットの表示モードを設定します。
- Curve Style(カーブスタイル)
- *Pattern(**パターン**)* - 曲線パターンを設定します。
- *Thickness(**太さ**)* - 曲線の太さを設定します。

Variable filter

- *Filter Interval* - 変数のフィルター処理の遅延。 遅延が必要ない場合は、値を 0 に設定します。

2.21.15 Figaro

- Figaro
- *Figaro Library(Figaro* **ライブラリ***)* - Figaro のライブラリパス。
- *Tree generation options(***ツリー生成操作***)* - フィガロツリー生成オプションのファイルパス。
- *Figaro Processor(Figaro* **プロセッサ***)* - フィガロのプロセッサの場所。

2.21.16 Debugger(デバガ)

- Algorithmic Debugger(アルゴリズムデバガ)
- *GDB Path(GDB* **パス***)* - GNU デバッガパス
- *GDB Command Timeout(GDB* **コマンド タイムアウト***)* - GDB コマンドのタイムアウト。
- *GDB Output Limit(GDB* **出力制限***)* - N 文字数に GDB の出力を制限します。
- *Display C frames(C* **のフレームを表示する***)* - true の場合、C スタックフレームを表示します。
- *Display unknown frames(***未知のフレームを表示する***)* - true の場合、不明なスタックフレームを表示します。 不明なスタックフレームとは、ファイルパスが不明なフレームを意味します。
- *Clear old output on a new run(***新規実行時に以前の出力をクリアする***)* - true の場合は、新しいシミュレーション実行時に出力ウィンドウをクリアします。
- *Clear old log on new run(***新規実行時に以前のログをクリアする***)* - true の場合、新しいシミュレーション実行時にログウィンドウをクリアします。
- Transformational Debugger(変換デバッガ)
- *Always show Transformational Debugger after compilation(***常にコンパイル後に変換デバッガを表示***)* - true の場合、モデルのコンパイル後に常に TransformationalDebugger ウィンドウを開きます。
- *Generate operations in the info xml(***操作を作る***)* - true の場合、infoxml ファイルに操作情報を追加します。

2.21.17 FMI

- Export
  - Version(バージョン)
  - *1.0* - FMI エクスポートのバージョンを 1.0 に設定します。
  - *2.0* - FMI エクスポートのバージョンを 2.0 に設定します。
  - Type(タイム)
  - *Model Exchange* - FMI エクスポートタイプをモデル交換に設定します。
  - *Co-Simulation* - FMI エクスポートタイプを Co-Simulation に設定します。
  - *Model Exchange and Co-Simulation* - FMI エクスポートタイプを ModelExchange と Co-Simulation に設定します。
  - *FMU* **名***:* - 生成した FMU ファイルの接頭辞を設定します。
  - Platforms - FMU のバイナリを生成するためのプラットフォームのリスト。
- Import
- *Delete FMU directory and generated model when OMEdit is closed* - true の場合、FMU をインポートするために作成された一時的な FMU ディレクトリが削除されます。

2.21.18 OMTLMSimulator

- General(全般)
- *Path(***パス***)* - OMTLMSimulator bin ディレクトリへのパス。
- **マネージャプロセス** - OMTLMSimulator managar プロセスへのパス。
- **モニタープロセス** - OMTLMSimulator モニタープロセスへのパス。

2.21.19 OMSimulator

- General(全般)
- *Working Directory(***作業ディレクトリ***)* - OMSimulator ファイル用の作業ディレクトリ。
- *Logging Level* - OMSimulator のログレベル。

2.22 __OpenModelica_commandLineOptions アノテーション

モデルのシミュレーションに必要なコマンドラインオプションを定義するための OpenModelica 固有のアノテーションです。 たとえば、デフォルトのアルゴリズムではなく、特定のマッチングアルゴリズムとインデックス低減方法を使用してモデルを常にシミュレートする場合は、次のコードを記述します。

```
model Test
  annotation(__OpenModelica_commandLineOptions = "--matchingAlgorithm=BFSB --
↪indexReductionMethod=dynamicStateSelection");
end Test;
```

アノテーションはスペースで区切られたオプションのリストであり、各オプションは単なるコマンドラインフラグまたは値を持つフラグのいずれかです。

OMEdit でこのオプションを設定する場合は、シミュレーションセットアップを開き変換フラグを設定してから、 *Save translation flags inside model i.e., __OpenModelica_commandLineOptions annotation* をオンにして、[OK] をクリックします。

このアノテーションを無効にしたい場合、 *setCommandLineOptions("--ignoreCommandLineOptionsAnnotation=true")* を使用してください。 OMEdit では **ツール > オプション > シミュレーション** から *Ignore __OpenModelica_commandLineOptions annotation* をチェックしてください。

2.23 __OpenModelica_simulationFlags アノテーション

モデルのシミュレーションに必要なシミュレーションオプションを定義するための OpenModelica 固有のアノテーションです。 たとえば、デフォルトの DASSL ソルバーの代わりに常に特定のソルバーを使用してモデルをシミュレートし、CPU 時間を確認したい場合は、次のコードを記述します。

```
model Test
  annotation(__OpenModelica_simulationFlags(s = "heun", cpu = "()"));
end Test;
```

アノテーションはコンマ区切りリストのオプションであり、各オプションは値を持つシミュレーションフラグです。 値を持たないフラグの場合は、 *()* を使用します（上記のコード例を参照）。

OMEdit で、シミュレーションセットアップを開き、シミュレーションフラグを設定します。次に、下部にある *__OpenModelica_simulationFlags annotation* チェックボックスをオンにします。

このアノテーションを無効にする場合は、 *setCommandLineOptions("--ignoreSimulationFlagsAnnotation=true")* を使用します。OMEdit から無効にするには、 *Tools > Options > Simulation* から *Ignore __OpenModelica_simulationFlags annotation* をチェックします。

2.24 Debugger(デバガ)

デバッグ機能に対しては、 *Debugging* を参照してください。

2.25 Modelica Standard Library の編集

デフォルトでは、OMEdit は Modelica Standard Library（MSL）をシステムライブラリとしてロードします。システムライブラリは読み取り専用です。 MSL を編集する場合は、システムライブラリではなくユーザーライブラリとしてロードする必要があります。 MSL を編集することはお勧めしませんが、本当に必要かつ内容を理解する必要がある場合は、次の手順に従ってください。

- *[ツール]*> *[オプション]*> *[ライブラリ]* に移動します。
- システムライブラリのリストから Modelica と ModelicaReference を削除します。
- *force loading of Modelica Standard Library* のチェックを外します。
- ユーザライブラリに *$OPENMODELICAHOME/lib/omlibrary/Modelica X.X/package.mo* を追加します。
- OMEdit を再起動します。

2.26 ステートマシン

2.26.1 Modelica ステートマシンの新規作成

Modelica クラスの新規作成 と同じ手順でモデルを作成します。そして *State* チェックボックスをチェックします。

2.26.2 遷移の作成

ある状態から別の状態に遷移するには、ユーザーは最初にツールバーから translation mode () を有効にする必要があります。

状態の上にマウスを移動します。 マウスカーソルが矢印カーソルからクロスカーソルに変わります。 トランジションを開始するには、左ボタンを押し、ボタンを押したまま移動します。 次に、左ボタンを放します。 終了状態に移動し、カーソルがクロスカーソルに変わったらクリックします。

Create Transition ダイアログボックスが表示され、トランジション属性を設定できます。 ダイアログをキャンセルすると、遷移がキャンセルされます。

トランジションをダブルクリックするか、右クリックして *Edit Transition* を選択し、トランジションの属性を変更します。

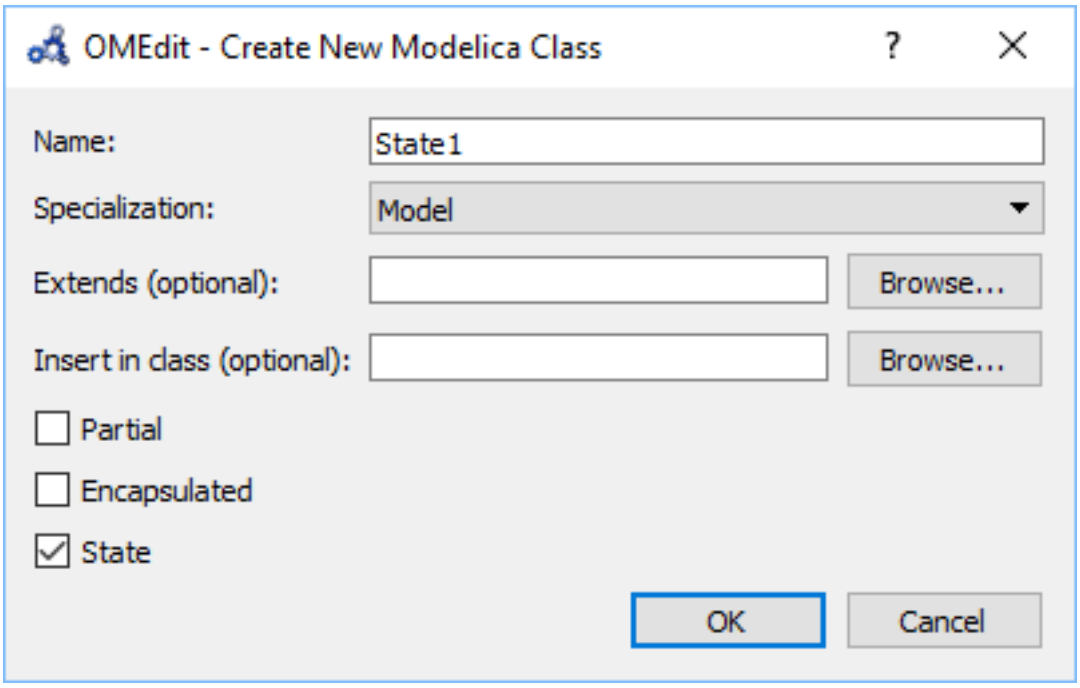


Figure 2.14: Modelica ステートマシンの作成

2.26.3 ステートマシンのミュレーション

Modelica ステートマシンのサポートは、Modelica Language Specification v3.3 で追加されました。 Modelica v3.2 ライブラリ（Modelica Standard Library v3.2.2 など）がロードされている場合、OMC が自動的に Modelica v3.2 互換モードに切り替わるため、微妙な問題が発生する可能性があります。 Modelica v3.2 互換モードでステートマシンをシミュレートしようとすると、エラーが発生します。（少なくとも）Modelica v3.3 に対応するためには OMC フラグ *--std=latest* を使用します。 OMEdit では *Tools* > *Options* > *Simulation* ダイアログでそのフラグを設定できます。

2.27 テキストエディタとして OMEdit を使用する

OMEdit はテキストエディタとして使用できます。 現在、MetaModelica、Modelica、C / C ++の編集のサポートとしては、構文ハイライトとキーワードとタイプのオートコンプリートが利用できます。 さらに、Modelica ファイルと MetaModelica ファイルには、キーワードとタイプとともにコードスニペットのオートコンプリートが用意されています。 ユーザーは、 *File* > *Open Directory* からディレクトリをロードできます。 これにより、ライブラリブラウザでディレクトリ構造が開きます。

ライブラリ・ブラウザでディレクトリが開かれた後、ユーザーはディレクトリ構造を展開し、ファイルをクリックしテキストエディタで開くことができます。

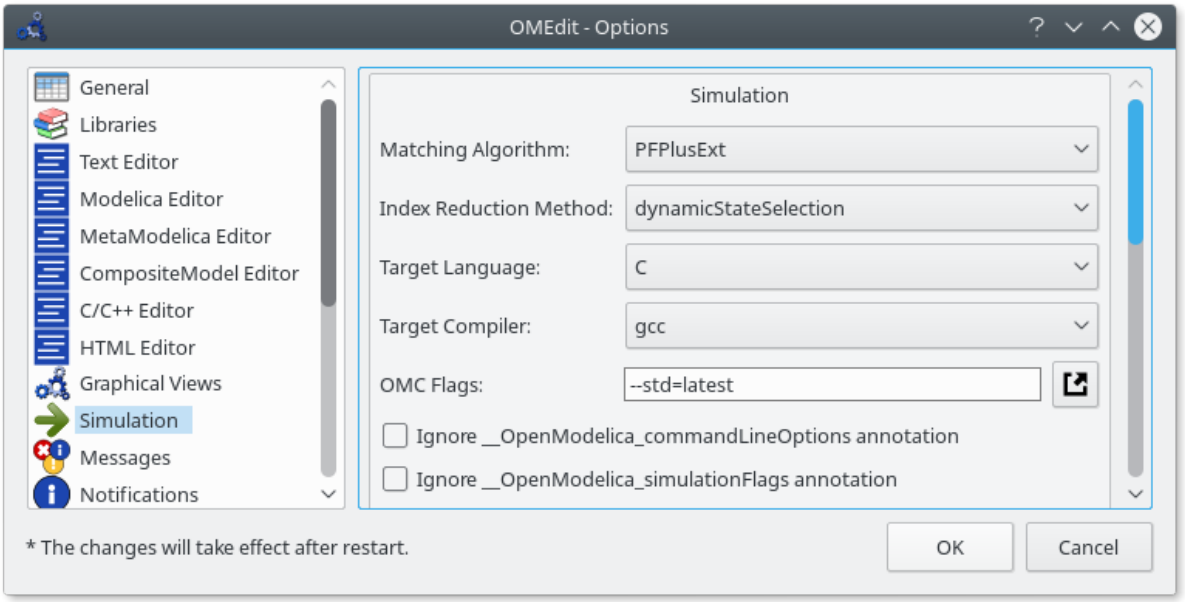


Figure 2.15: （少なくとも）Modelica v3.3 でのサポートを確認

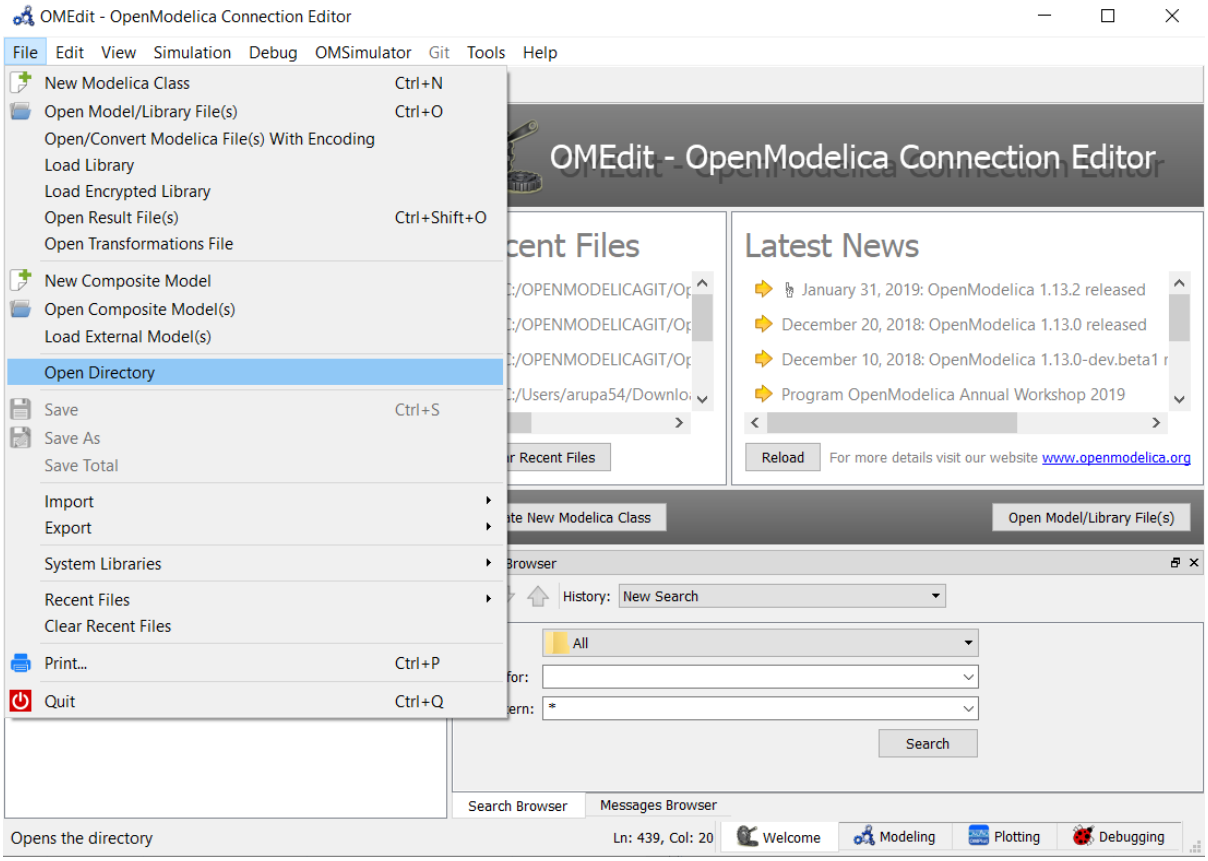


Figure 2.16: ディレクトリを開く

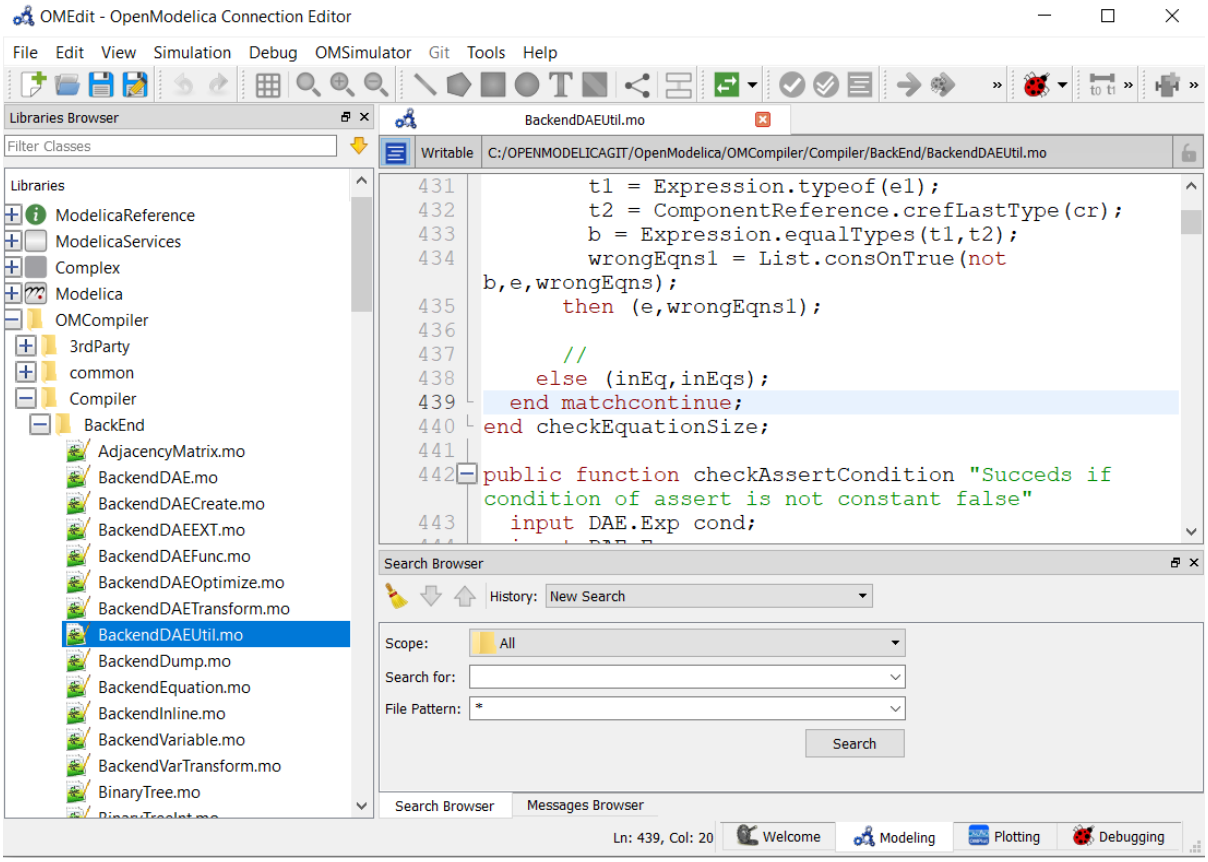


Figure 2.17: テキストエディタでファイルを開く

2.27.1 高度な検索

OMEdit テキストエディタでは検索機能が利用可能です。 検索ブラウザを有効にするには、[ビュー]> [ウィンドウ]> [Search browser] を選択するか、ショートカットキー（Ctrl + H）を使用します。

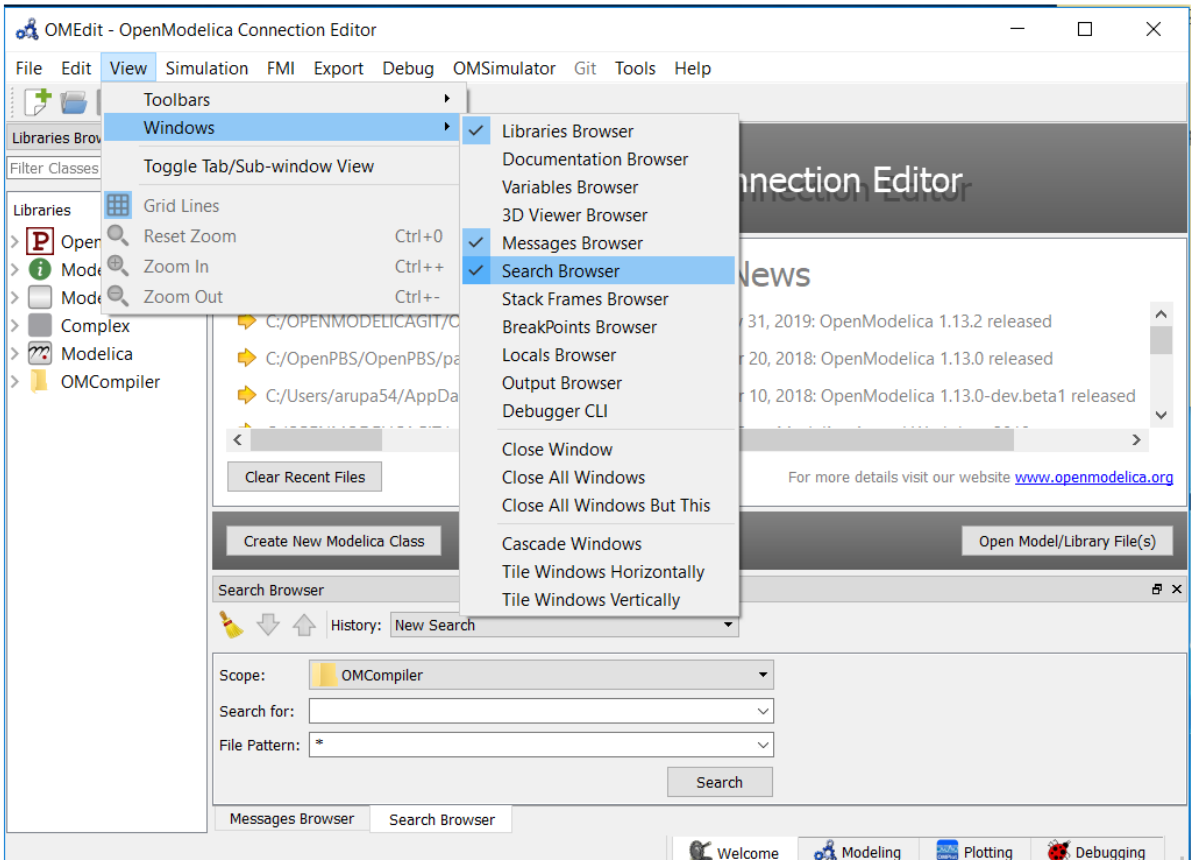


Figure 2.18: OMEdit search browser を有効化

ユーザーは、検索したいディレクトリをロードし、検索するテキストと必要に応じてファイルパターンを入力して検索ボタンをクリックすることで、検索を開始できます。

検索が完了すると、結果は別のウィンドウで表示されます。検索結果には以下が含まれます。

1) 検索ワードが一致したファイルの名前

2) 一致した単語の行番号とテキスト

ユーザーが行番号または一致したテキストをクリックすると、テキストエディタでファイルが自動的に開き、テキストが一致した行番号にカーソルが移動します。

ユーザーが複数の検索を実行し、検索履歴オプションを使用して、古い検索結果に戻ることができます。

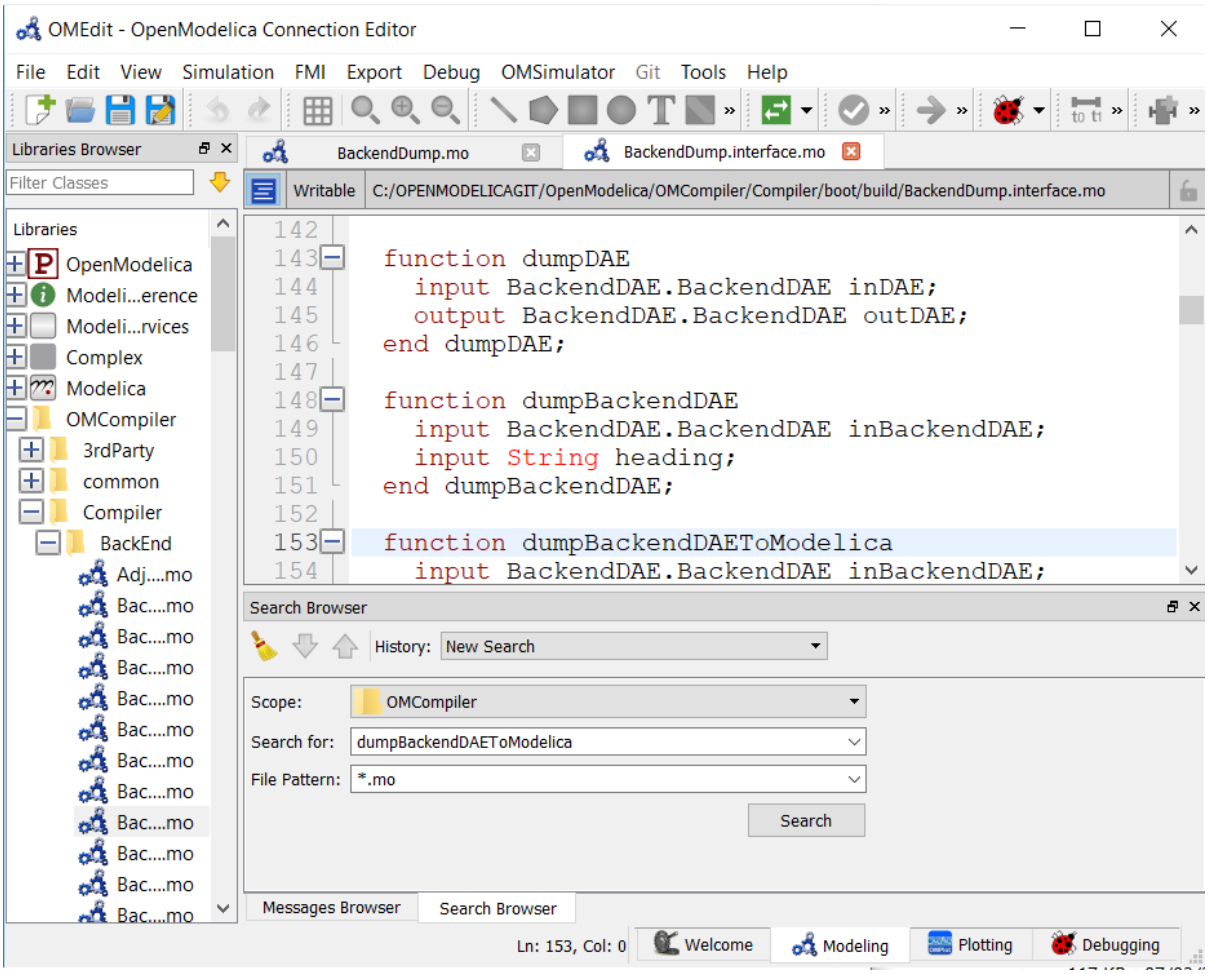


Figure 2.19: search browser で検索開始

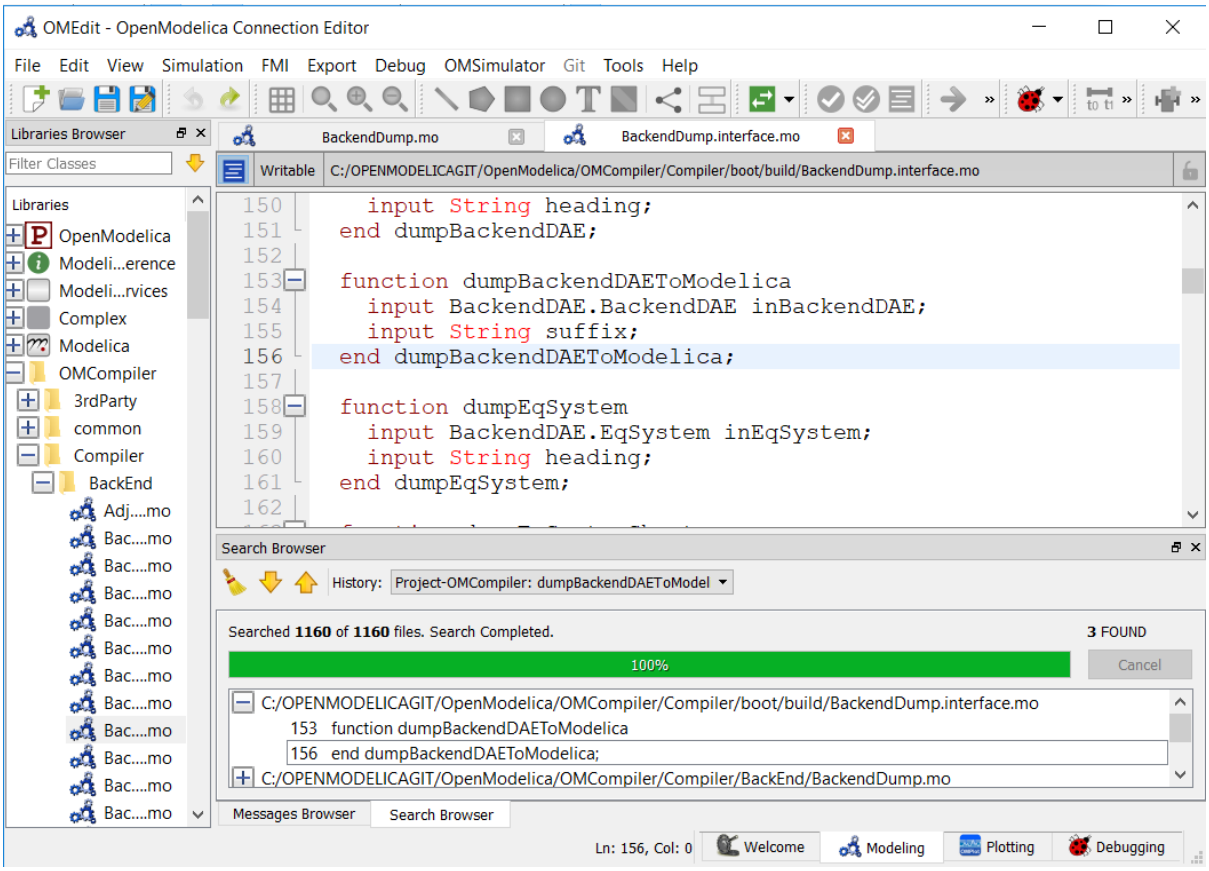


Figure 2.20: 検索結果

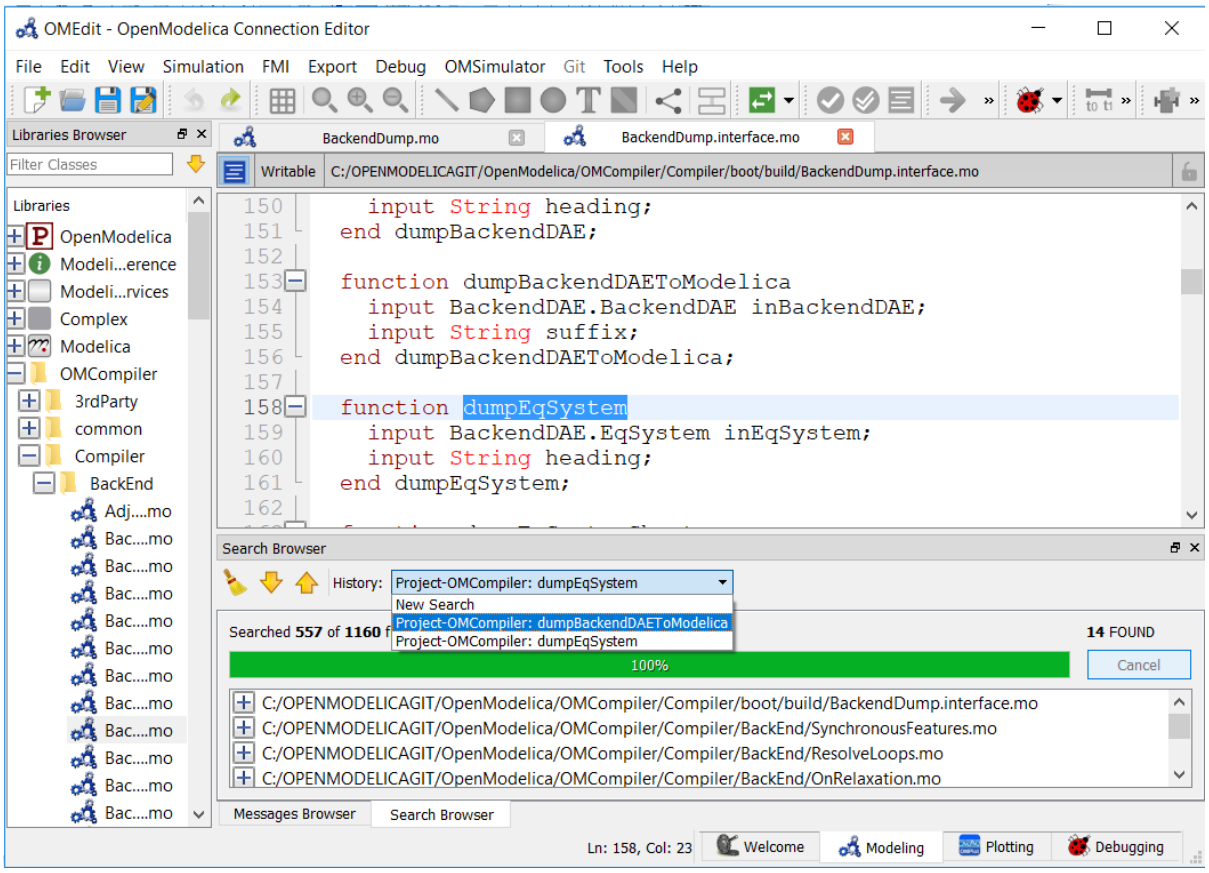


Figure 2.21: 検索履歴

第3章　2Dプロット

この章では、OMNotebook、OMShell、およびコマンドラインスクリプトを介して OpenModelica で使用できる 2D プロットについて説明します。 プロットは OMPlot アプリケーションに基づいています。

3.1 サンプル

```
class HelloWorld
  Real x(start = 1, fixed = true);
  parameter Real a = 1;
equation
  der(x) = - a * x;
end HelloWorld;
```

簡単な時間プロットを作成するために、上記のモデル HelloWorld を計算します。 この例でシミュレーションデータの量を減らすために、出力結果のプロット数は引数 numberOfIntervals = 5 で制限されています。 シミュレーションは、以下のコマンドで開始されます。

```
>>> simulate(HelloWorld, outputFormat="csv", startTime=0, stopTime=4,␣
↪numberOfIntervals=5)
record SimulationResult
    resultFile = "<<DOCHOME>>/HelloWorld_res.csv",
    simulationOptions = "startTime = 0.0, stopTime = 4.0, numberOfIntervals = 5,
 tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'HelloWorld', options = '',␣
↪outputFormat = 'csv', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.0149018,
    timeBackend = 0.01952009999999993,
    timeSimCode = 0.0016015,
    timeTemplates = 0.0129376,
    timeCompile = 2.5339383,
    timeSimulation = 0.1132089,
    timeTotal = 2.6984581
end SimulationResult;
```

シミュレーションが終了すると、ファイル *HelloWorld_res.csv* にシミュレーションデータが含まれます。

Listing 3.1: HelloWorld_res.csv

```
"time","x","der(x)"
0,1,-1
0.8,0.4493289092712475,-0.4493289092712475
1.6,0.2018973974273906,-0.2018973974273906
2.4,0.09071896372718975,-0.09071896372718975
```

(次のページに続く)

(前のページからの続き)

```
3.2,0.04076293845066793,-0.04076293845066793
4,0.01831609502171534,-0.01831609502171534
4,0.01831609502171534,-0.01831609502171534
```

次の plot コマンドを使用して、新しい OMPlot プログラムを用いてプロットが作成されます。

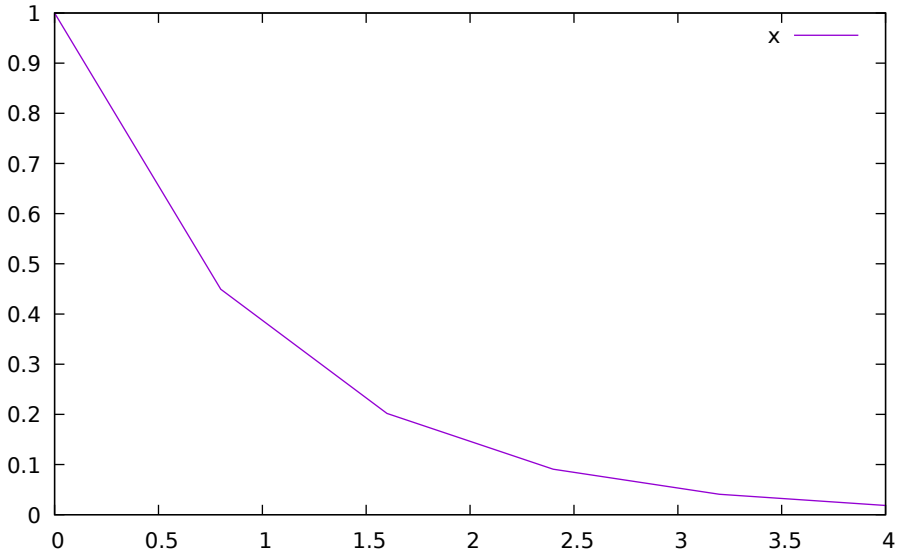


Figure 3.1: HelloWorld の簡単な 2D プロット例

例えばデフォルトの 500 間隔を使用してより多くのポイントで再シミュレーションして結果を保存することにより、スムーズなプロットを取得できます

```
>>> 0==system("./HelloWorld -override stepSize=0.008")
true
>>> res:=strtok(readFile("HelloWorld_res.csv"), "\n");
>>> res[end]
"4,0.01831609502171534,-0.01831609502171534"
```

3.2 プロットコマンドインターフェース

plot コマンドは結果の図をカスタマイズするためのオプションがあります

```
>>> list(OpenModelica.Scripting.plot,interfaceOnly=true)
"function plot
  input VariableNames vars \"The variables you want to plot\";
  input Boolean externalWindow = false \"Opens the plot in a new plot window\";
  input String fileName = \"<default>\" \"The filename containing the variables.
↪<default> will read the last simulation result\";
  input String title = \"\" \"This text will be used as the diagram title.\";
  input String grid = \"detailed\" \"Sets the grid for the plot i.e simple,␣
↪detailed, none.\";
  input Boolean logX = false \"Determines whether or not the horizontal axis is␣
↪logarithmically scaled.\";
  input Boolean logY = false \"Determines whether or not the vertical axis is␣
↪logarithmically scaled.\";
```

(次のページに続く)

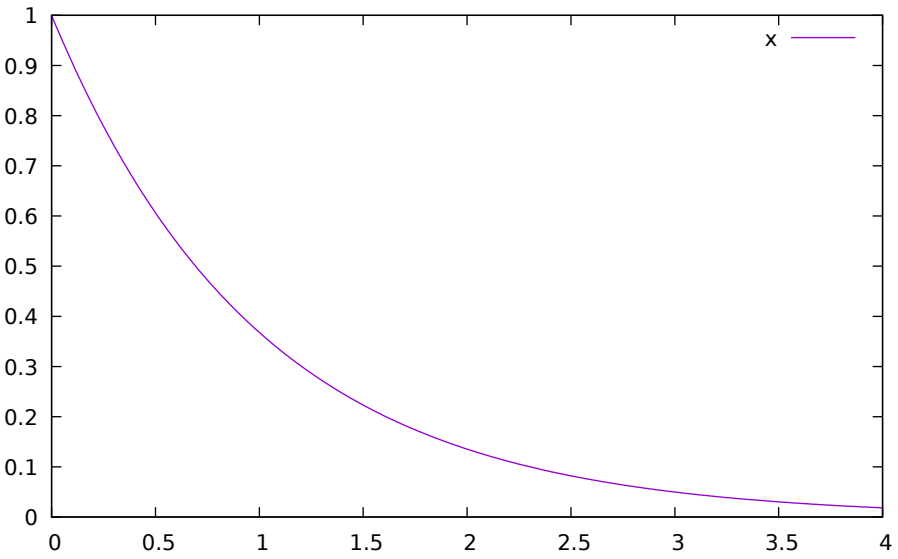


Figure 3.2: 多くの出力点を用いた HelloWorld の 2D プロット例

(前のページからの続き)

```
  input String xLabel = \"time\" \"This text will be used as the horizontal label␣
↪in the diagram.\";
  input String yLabel = \"\" \"This text will be used as the vertical label in the␣
↪diagram.\";
  input Real xRange[2] = {0.0, 0.0} \"Determines the horizontal interval that is␣
↪visible in the diagram. {0,0} will select a suitable range.\";
  input Real yRange[2] = {0.0, 0.0} \"Determines the vertical interval that is␣
↪visible in the diagram. {0,0} will select a suitable range.\";
  input Real curveWidth = 1.0 \"Sets the width of the curve.\";
  input Integer curveStyle = 1 \"Sets the style of the curve. SolidLine=1,␣
↪DashLine=2, DotLine=3, DashDotLine=4, DashDotDotLine=5, Sticks=6, Steps=7.\";
  input String legendPosition = \"top\" \"Sets the POSITION of the legend i.e left,
 right, top, bottom, none.\";
  input String footer = \"\" \"This text will be used as the diagram footer.\";
  input Boolean autoScale = true \"Use auto scale while plotting.\";
  input Boolean forceOMPlot = false \"if true launches OMPlot and doesn't call␣
↪callback function even if it is defined.\";
  output Boolean success \"Returns true on success\";
end plot;"
```

第4章　Solving Modelica Models

4.1 Integration Methods

By default OpenModelica transforms a Modelica model into an ODE representation to perform a simulation by using numerical integration methods. This section contains additional information about the different integration methods in OpenModelica. They can be selected by the method parameter of the *simulate* command or the *-s simflag*.

The different methods are also called solver and can be distinguished by their characteristic:

- explicit vs. implicit
- order
- step size control
- multi step

A good introduction on this topic may be found in [CK06] and a more mathematical approach can be found in [HNorsettW93].

4.1.1 DASSL

DASSL is the default solver in OpenModelica, because of a severals reasons. It is an implicit, higher order, multi-step solver with a step-size control and with these properties it is quite stable for a wide range of models. Furthermore it has a mature source code, which was originally developed in the eighties an initial description may be found in [Pet82].

This solver is based on backward differentiation formula (BDF), which is a family of implicit methods for numerical integration. The used implementation is called DASPK2.0 (see[2]) and it is translated automatically to C by f2c (see[3]).

The following simulation flags can be used to adjust the behavior of the solver for specific simulation problems: *jacobian*, *noRootFinding*, *noRestart*, *initialStepSize*, *maxStepSize*, *maxIntegrationOrder*, *noEquidistantTimeGrid*.

[2] DASPK Webpage
[3] Cdaskr source

4.1.2 IDA

The IDA solver is part of a software family called sundials: SUite of Nonlinear and DIfferential/ALgebraic equation Solvers [HBG+05]. The implementation is based on DASPK with an extended linear solver interface, which includes an interface to the high performance sparse linear solver KLU [DN10].

The simulation flags of *DASSL* are also valid for the IDA solver and furthermore it has the following IDA specific flags: *idaLS*, *idaMaxNonLinIters*, *idaMaxConvFails*, *idaNonLinConvCoef*, *idaMaxErrorTestFails*.

4.1.3 Basic Explicit Solvers

The basic explicit solvers are performing with a fixed step-size and differ only in the integration order. The step-size is based on the numberOfIntervals, the startTime and stopTime parameters in the *simulate* command: $\text{stepSize} \approx \frac{\text{stopTime} - \text{startTime}}{\text{numberOfIntervals}}$

- euler - order 1
- heun - order 2
- rungekutta - order 4

4.1.4 Basic Implicit Solvers

The basic implicit solvers are all based on the non-linear solver KINSOL from the SUNDIALS suite. The underlining linear solver can be modified with the simflag *-impRKLS*. The step-size is determined as for the basic explicit solvers.

- impeuler - order 1
- trapezoid - order 2
- imprungekutta - Based on Radau IIA and Lobatto IIIA defined by its Butcher tableau where the order can be adjusted by *-impRKorder*.

4.1.5 Experimental Solvers

The following solvers are marked as experimental, mostly because they are till now not tested very well.

- rungekuttaSsc - Runge-Kutta based on Novikov (2016) - explicit, step-size control, order 4-5
- irksco - Own developed Runge-Kutta solver - implicit, step-size control, order 1-2
- symSolver - Symbolic inline solver (requires *--symSolver*) - fixed step-size, order 1
- symSolverSsc - Symbolic implicit inline Euler with step-size control (requires *--symSolver*) - step-size control, order 1-2
- qss - A QSS solver

4.2 DAE Mode Simulation

Beside the default ODE simulation, OpenModelica is able to simulate models in *DAE mode*. The *DAE mode* is enabled by the flag *--daeMode*. In general the whole equation system of a model is passed to the DAE integrator, including all algebraic loops. This reduces the amount of work that needs to be done in the post optimization phase of the OpenModelica backend. Thus models with large algebraic loops might compile faster in *DAE mode*.

Once a model is compiled in *DAE mode* the simulation can be only performed with *SUNDIALS/IDA* integrator and with enabled *-daeMode* simulation flag. Both are enabled automatically by default, when a simulation run is started.

4.2.1 References

第5章　Debugging

There are two main ways to debug Modelica code, the transformations browser, which shows the transformations OpenModelica performs on the equations. There is also a debugger for *debugging of algorithm sections and functions*.

5.1 The Equation-based Debugger

This section gives a short description how to get started using the equation-based debugger in OMEdit.

5.1.1 Enable Tracing Symbolic Transformations

This enables tracing symbolic transformations of equations. It is optional but strongly recommended in order to fully use the debugger. The compilation time overhead from having this tracing on is less than 1%, however, in addition to that, some time is needed for the system to write the xml file containing the transformation tracing information.

Enable -d=infoXmlOperations in Tools->Options->Simulation (see section シミュレーション) OR alternatively click on the checkbox *Generate operations in the info xml* in Tools->Options->Debugger (see section *Debugger(デバガ)*) which performs the same thing.

This adds all the transformations performed by OpenModelica on the equations and variables stored in the model_info.xml file. This is necessary for the debugger to be able to show the whole path from the source equation(s) to the position of the bug.

5.1.2 Load a Model to Debug

Load an interesting model. We will use the package Debugging.mo since it contains suitable, broken models to demonstrate common errors.

5.1.3 Simulate and Start the Debugger

Select and simulate the model as usual. For example, if using the Debugging package, select the model Debugging.Chattering.ChatteringEvents1. If there is an error, you will get a clickable link that starts the debugger. If the user interface is unresponsive or the running simulation uses too much processing power, click cancel simulation first.

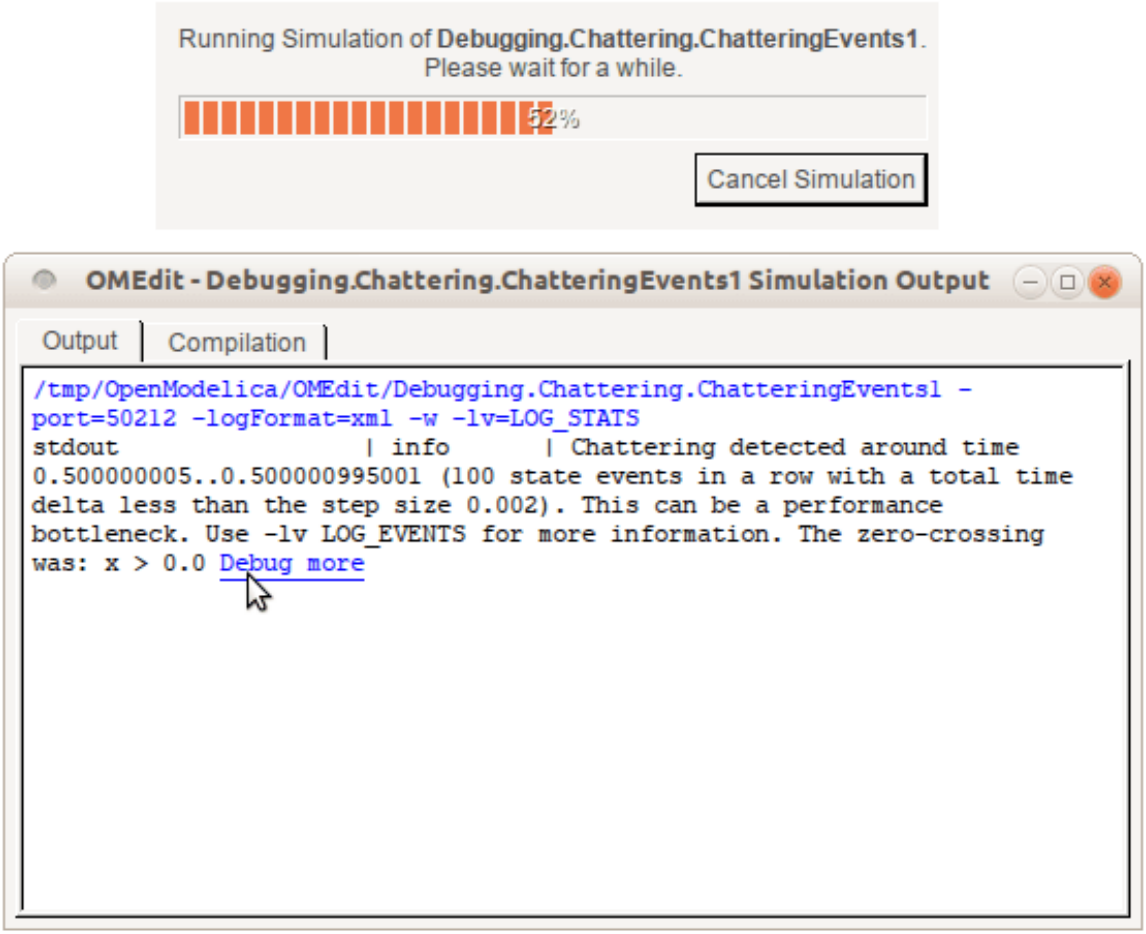


Figure 5.1: Simulating the model.

5.1.4 Use the Transformation Debugger for Browsing

Use the transformation debugger. It opens on the equation where the error was found. You can browse through the dependencies (variables that are defined by the equation, or the equation is dependent on), and similar for variables. The equations and variables form a bipartite graph that you can walk.

If the -d=infoXmlOperations was used or you clicked the “generate operations” button, the operations performed on the equations and variables can be viewed. In the example package, there are not a lot of operations because the models are small.

Try some larger models, e.g. in the MultiBody library or some other library, to see more operations with several transformation steps between different versions of the relevant equation(s). If you do not trigger any errors in a

model, you can still open the debugger, using File->Open Transformations File (model_info.json).

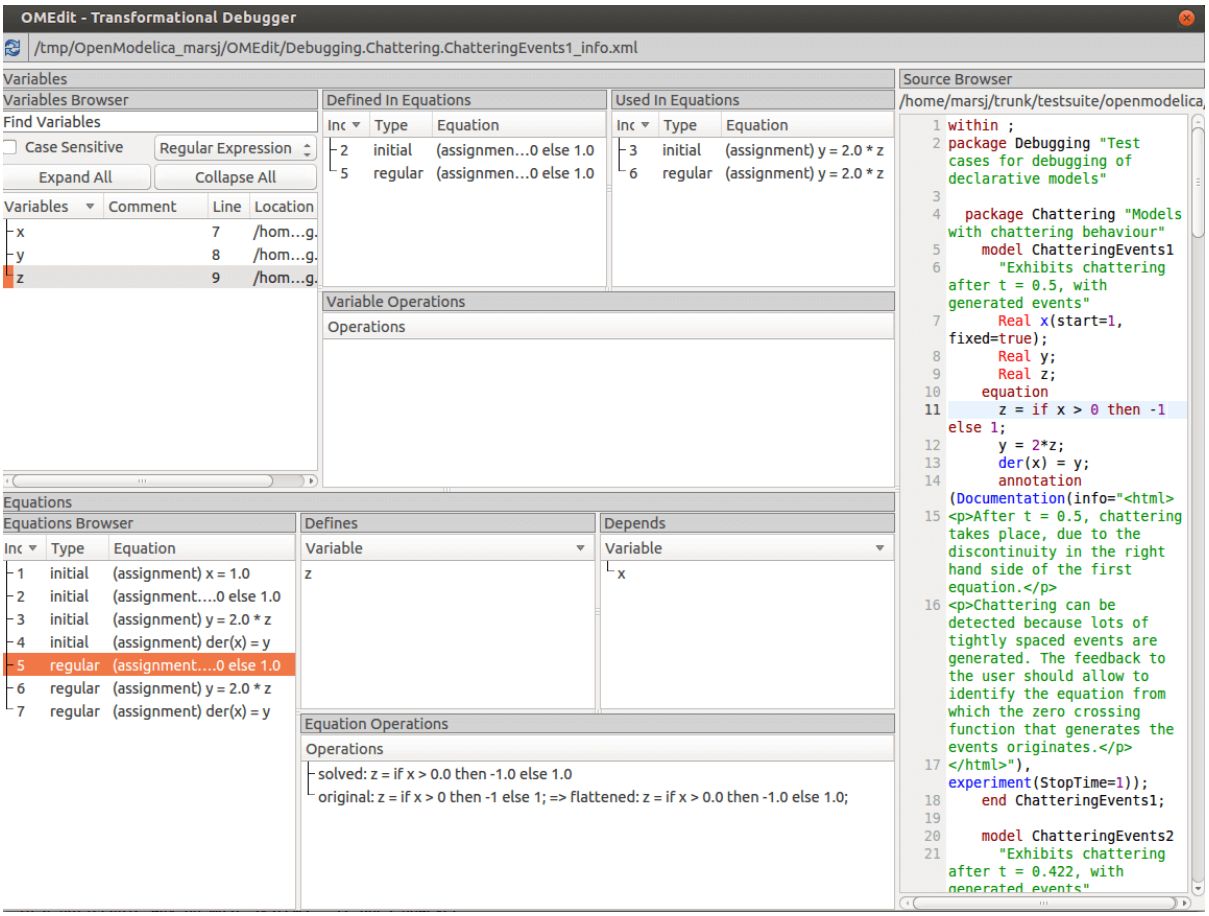


Figure 5.2: Transfomations Browser.

5.2 The Algorithmic Debugger

This section gives a short description how to get started using the algorithmic debugger in OMEdit. See section シミュレーション for further details of debugger options.

5.2.1 Adding Breakpoints

There are two ways to add the breakpoints,

- Click directly on the line number in Text View, a red circle is created indicating a breakpoint as shown in Figure 5.3.
- Open the Algorithmic Debugger window and add a breakpoint using the right click menu of Breakpoints Browser window.

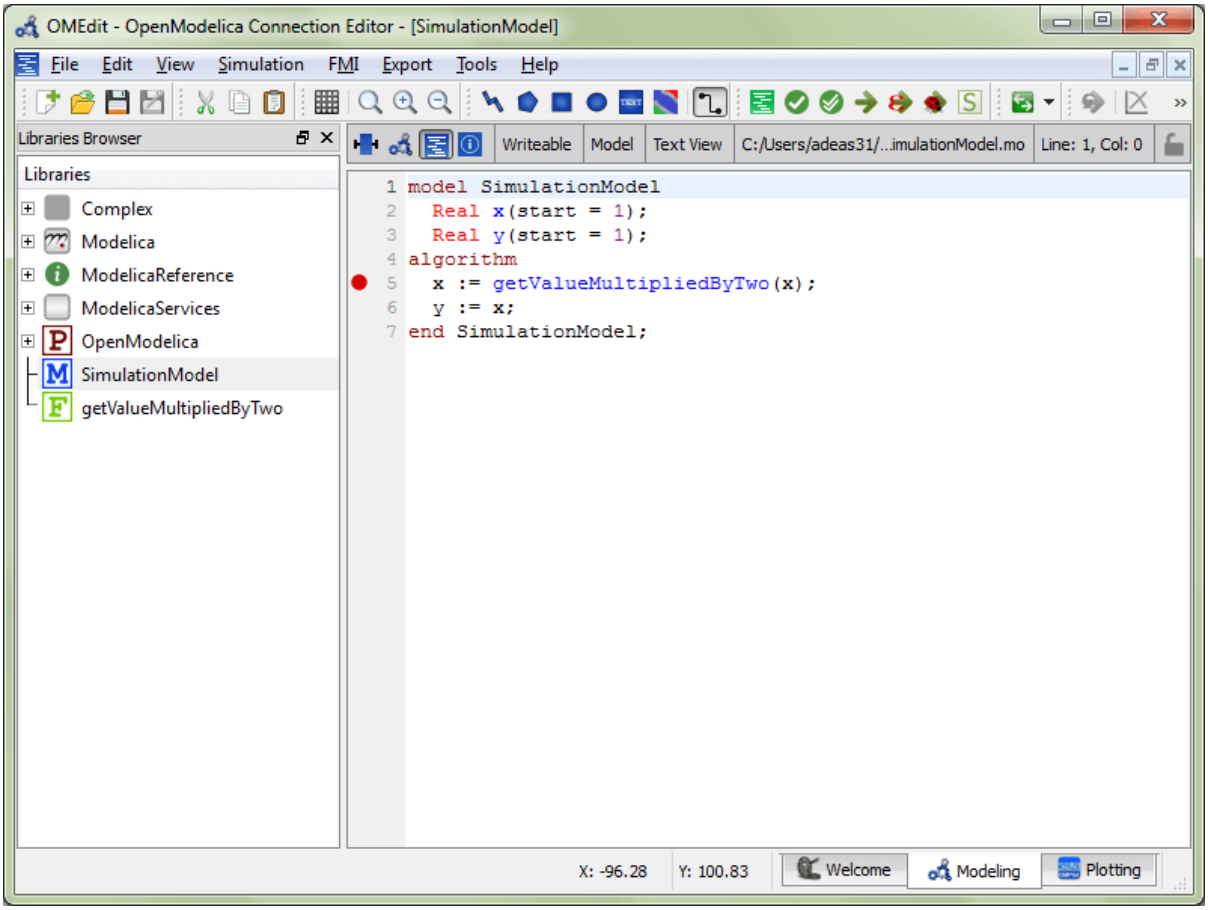


Figure 5.3: Adding breakpoint in Text View.

5.2.2 Start the Algorithmic Debugger

You should add breakpoints before starting the debugger because sometimes the simulation finishes quickly and you won't get any chance to add the breakpoints.

There are four ways to start the debugger,

- Open the Simulation Setup and click on Launch Algorithmic Debugger before pressing Simulate.
- Right click the model in Libraries Browser and select Simulate with Algorithmic Debugger.
- Open the Algorithmic Debugger window and from menu select Debug-> *Debug Configurations*.
- Open the Algorithmic Debugger window and from menu select Debug-> *Attach to Running Process*.

5.2.3 Debug Configurations

If you already have a simulation executable with debugging symbols outside of OMEdit then you can use the Debug->Debug Configurations option to load it.

The debugger also supports MetaModelica data structures so one can debug omc executable. Select omc executable as program and write the name of the mos script file in Arguments.

5.2.4 Attach to Running Process

If you already have a running simulation executable with debugging symbols outside of OMEdit then you can use the Debug->Attach to Running Process option to attach the debugger with it. Figure 5.5 shows the Attach to Running Process dialog. The dialog shows the list of processes running on the machine. The user selects the program that he/she wish to debug. OMEdit debugger attaches to the process.

5.2.5 Using the Algorithmic Debugger Window

Figure 5.6 shows the Algorithmic Debugger window. The window contains the following browsers,

- *Stack Frames Browser* – shows the list of frames. It contains the program context buttons like resume, interrupt, exit, step over, step in, step return. It also contains a threads drop down which allows switching between different threads.
- *BreakPoints Browser* – shows the list of breakpoints. Allows adding/editing/removing breakpoints.
- *Locals Browser* – Shows the list of local variables with values. Select the variable and the value will be shown in the bottom right window. This is just for convenience because some variables might have long values.
- *Debugger CLI* – shows the commands sent to gdb and their responses. This is for advanced users who want to have more control of the debugger. It allows sending commands to gdb.
- *Output Browser* – shows the output of the debugged executable.

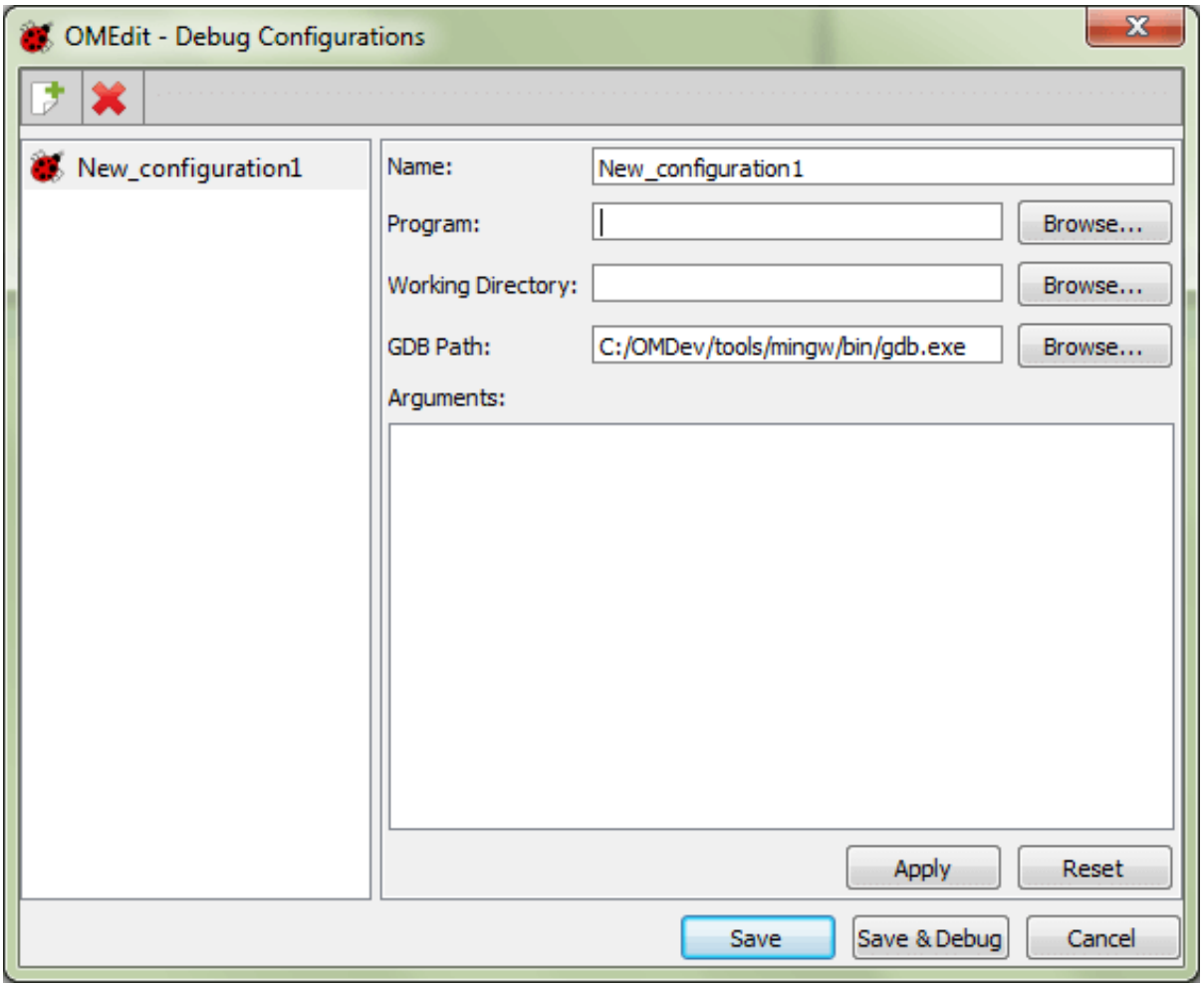


Figure 5.4: Debug Configurations.

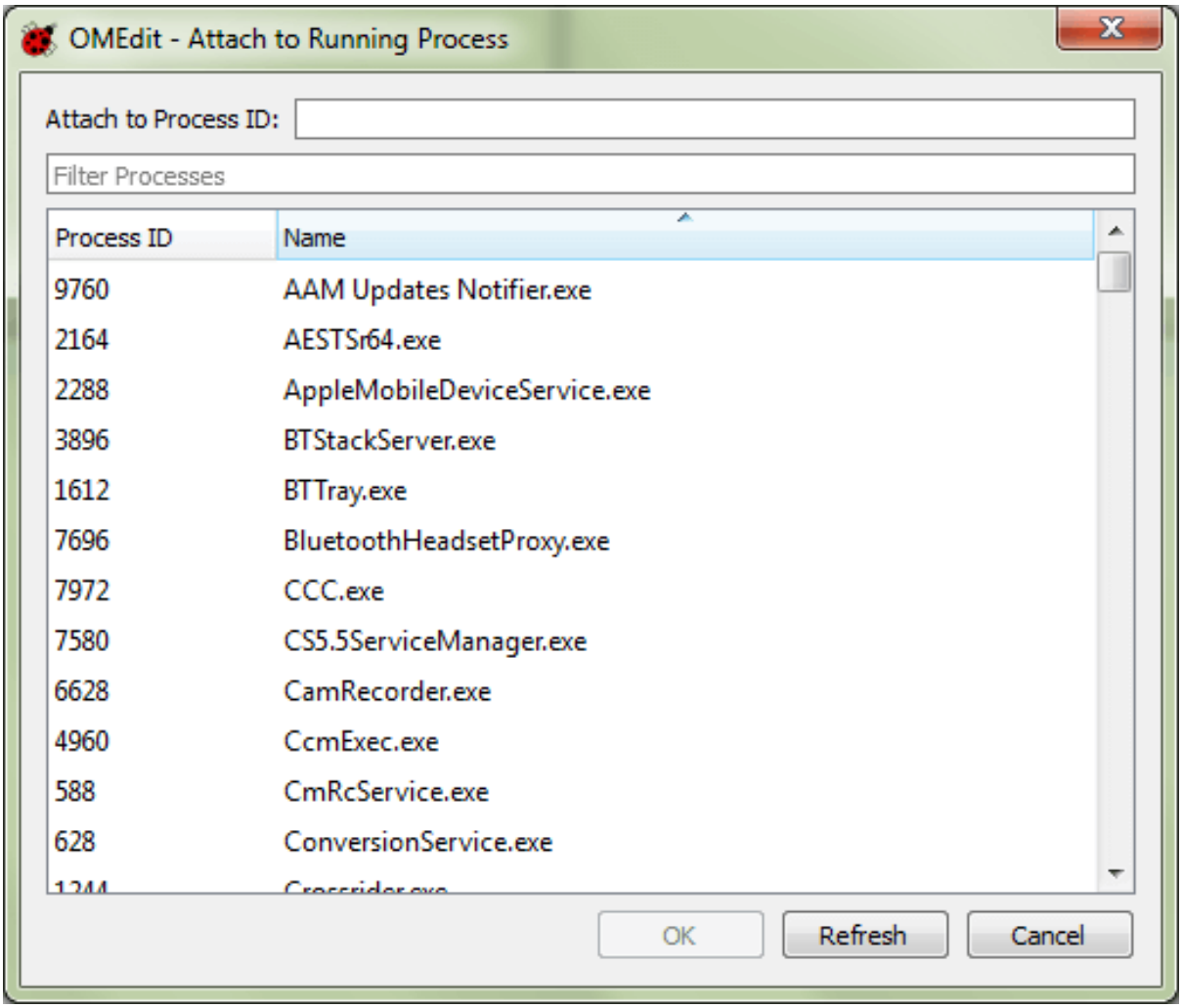


Figure 5.5: Attach to Running Process.

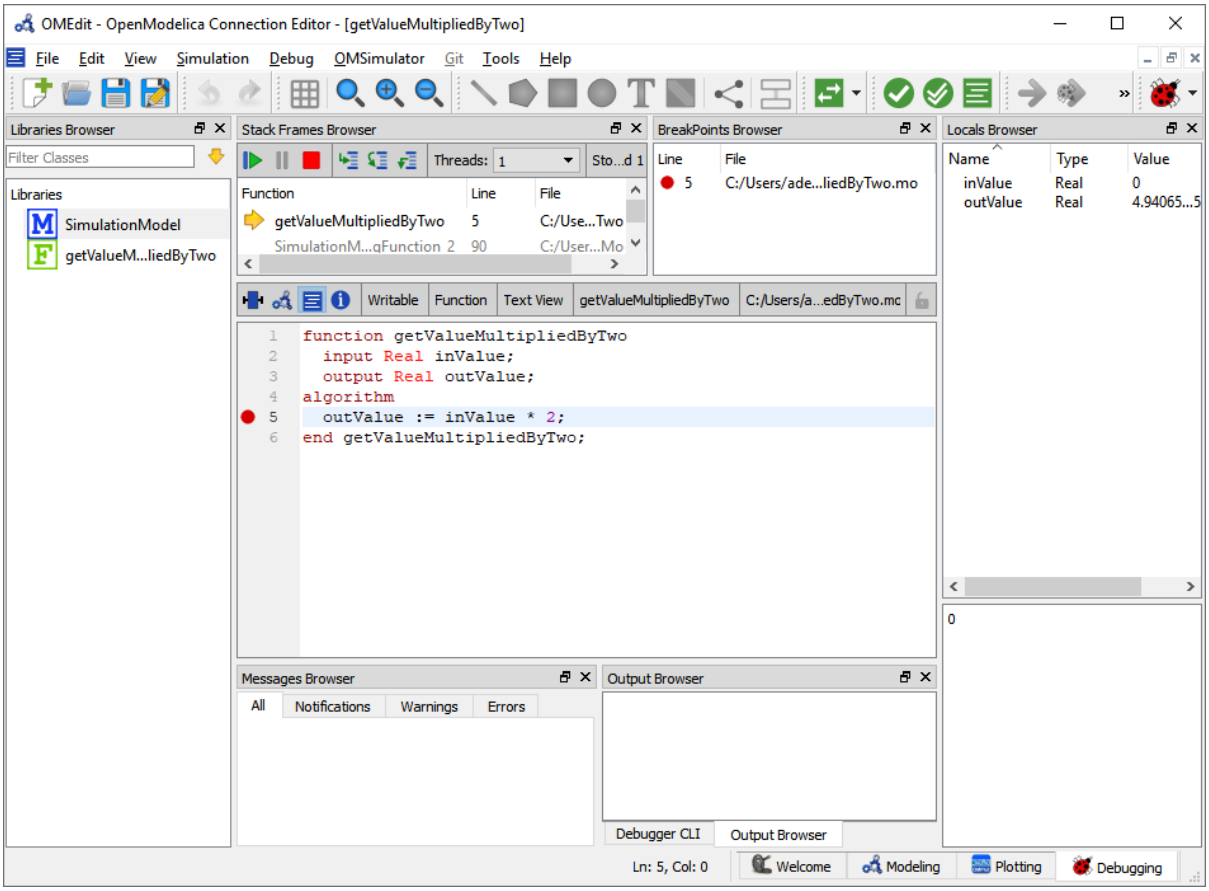


Figure 5.6: Algorithmic Debugger.

第6章 Generating Graph Representations for Models

The system of equations after symbolic transformation is represented by a graph. OpenModelica can generate graph representations which can be displayed in the graph tool *yed* (http://www.yworks.com/products/yed). The graph generation is activated with the debug flag

+d=graphml

Two different graphml- files are generated in the working directory. *TaskGraph_model.graphml*, showing the strongly-connected components of the model and *BipartiteGraph_CompleteDAE_model.graphml* showing all variables and equations. When loading the graphs with *yEd*, all nodes are in one place. Please use the various layout algorithms to get a better overview.

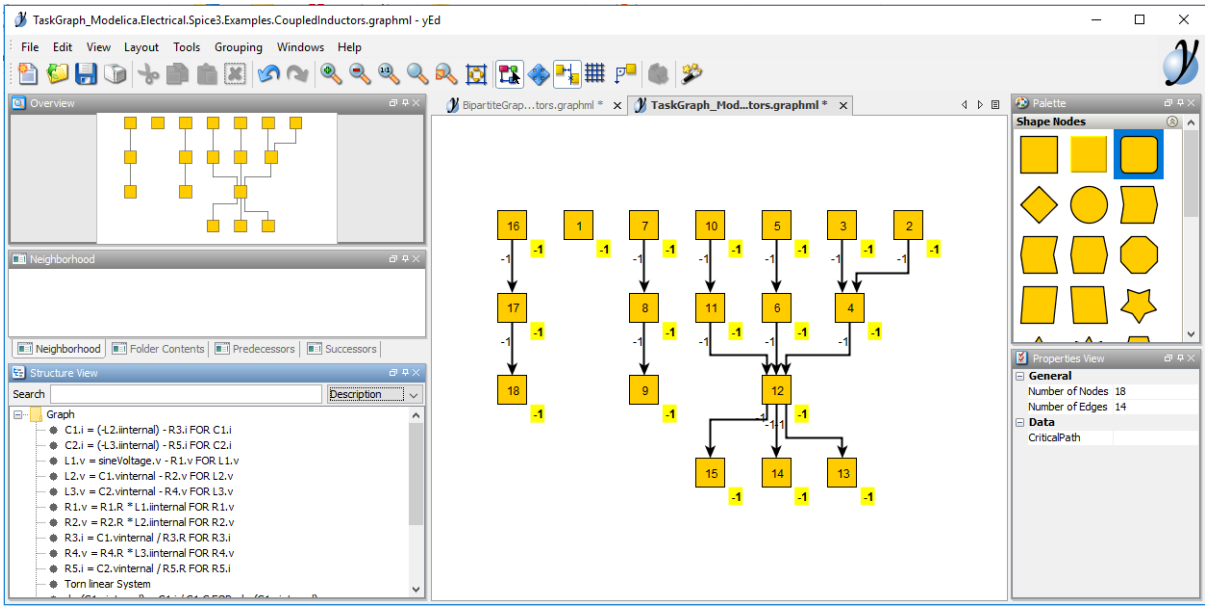


Figure 6.1: A task-graph representation of a model in yEd

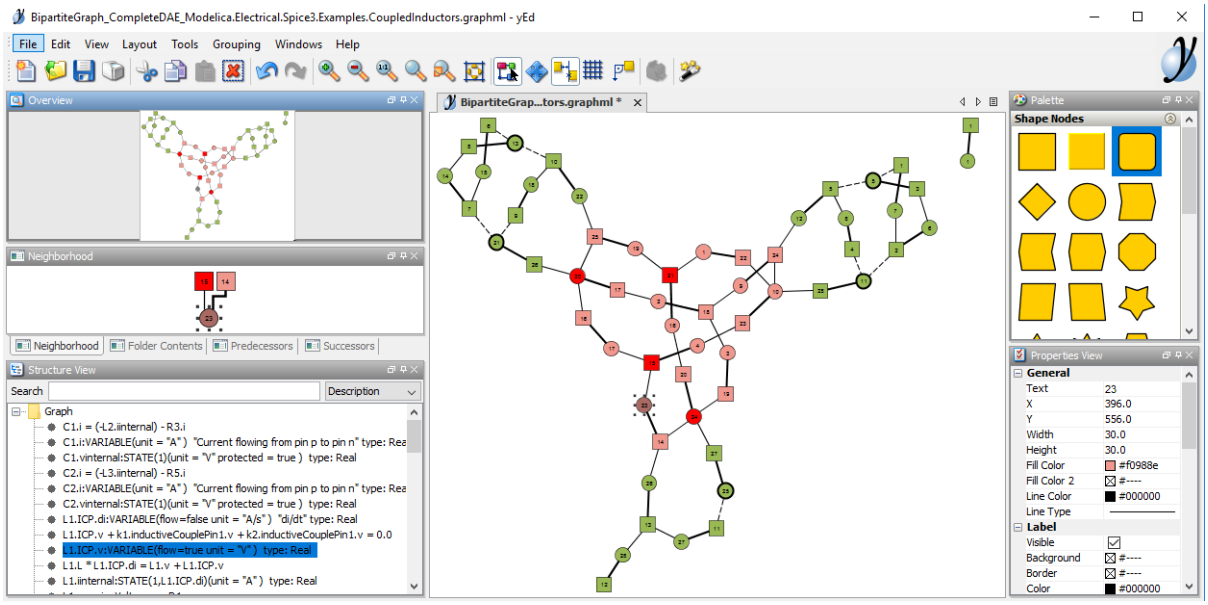


Figure 6.2: A biparite graph representation of a model in yEd

第7章　FMI and TLM-Based Simulation and Co-simulation of External Models

7.1 Functional Mock-up Interface - FMI

The new standard for model exchange and co-simulation with Functional Mockup Interface (FMI) allows export of pre-compiled models, i.e., C-code or binary code, from a tool for import in another tool, and vice versa. The FMI standard is Modelica independent. Import and export works both between different Modelica tools, or between certain non-Modelica tools. OpenModelica supports FMI 1.0 & 2.0,

- Model Exchange
- Co-Simulation (under development)

7.1.1 FMI Export

To export the FMU use the OpenModelica command translateModelFMU(ModelName) from command line interface, OMShell, OMNotebook or MDT. The export FMU command is also integrated with OMEdit. Select FMI > Export FMU the FMU package is generated in the current directory of omc. You can use the cd() command to see the current location. You can set which version of FMI to export through OMEdit settings, see section *FMI*.

To export the bouncing ball example to an FMU, use the following commands:

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/
↪BouncingBall.mo")
true
>>> translateModelFMU(BouncingBall)
"<<DOCHOME>>/BouncingBall.fmu"
>>> system("unzip -l BouncingBall.fmu | egrep -v 'sources|files' | tail -n+3 |␣
↪grep -o '[A-Za-z._0-9/]*$' > BB.log")
0
```

After the command execution is complete you will see that a file BouncingBall.fmu has been created. Its contents varies depending on the current platform. On the machine generating this documentation, the contents in Listing 7.1 are generated (along with the C source code).

Listing 7.1: BouncingBall FMU contents

```
binaries/
binaries/linux64/
binaries/linux64/BouncingBall.so
binaries/linux64/BouncingBall_FMU.libs
modelDescription.xml
```

A log file for FMU creation is also generated named ModelName_FMU.log. If there are some errors while creating FMU they will be shown in the command line window and logged in this log file as well.

By default an FMU that can be used for both Model Exchange and Co-Simulation is generated. We only support FMI 2.0 for Co-Simulation FMUs.

Currently the Co-Simulation FMU supports only the forward Euler solver with root finding which does an Euler step of communicationStepSize in fmi2DoStep. Events are checked for before and after the call to fmi2GetDerivatives.

7.1.2 FMI Import

To import the FMU package use the OpenModelica command importFMU,

```
>>> list(OpenModelica.Scripting.importFMU, interfaceOnly=true)
function importFMU
  input String filename "the fmu file name";
  input String workdir = "<default>" "The output directory for imported FMU files.
↪<default> will put the files to current working directory.";
  input Integer loglevel = 3 "loglevel_nothing=0;loglevel_fatal=1;loglevel_error=2;
↪loglevel_warning=3;loglevel_info=4;loglevel_verbose=5;loglevel_debug=6";
  input Boolean fullPath = false "When true the full output path is returned␣
↪otherwise only the file name.";
  input Boolean debugLogging = false "When true the FMU's debug output is printed.
↪";
  input Boolean generateInputConnectors = true "When true creates the input␣
↪connector pins.";
  input Boolean generateOutputConnectors = true "When true creates the output␣
↪connector pins.";
  output String generatedFileName "Returns the full path of the generated file.";
end importFMU;
```

The command could be used from command line interface, OMShell, OMNotebook or MDT. The importFMU command is also integrated with OMEdit. Select FMI > Import FMU the FMU package is extracted in the directory specified by workdir, since the workdir parameter is optional so if its not specified then the current directory of omc is used. You can use the cd() command to see the current location.

The implementation supports FMI for Model Exchange 1.0 & 2.0 and FMI for Co-Simulation 1.0 stand-alone. The support for FMI Co-Simulation is still under development.

The FMI Import is currently a prototype. The prototype has been tested in OpenModelica with several examples. It has also been tested with example FMUs from FMUSDK and Dymola. A more fullfleged version for FMI Import will be released in the near future.

When importing the model into OMEdit, roughly the following commands will be executed:

```
>>> imported_fmu_mo_file:=importFMU("BouncingBall.fmu")
"BouncingBall_me_FMU.mo"
>>> loadFile(imported_fmu_mo_file)
true
```

The imported FMU can then be simulated like any normal model:

```
>>> simulate(BouncingBall_me_FMU, stopTime=3.0)
record SimulationResult
    resultFile = "<<DOCHOME>>/BouncingBall_me_FMU_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 3.0, numberOfIntervals = 500,␣
↪tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'BouncingBall_me_FMU',␣
↪options = '', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags␣
↪= ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.04660130000000001,
    timeBackend = 0.0173985,
    timeSimCode = 0.0327956,
    timeTemplates = 0.0180952,
    timeCompile = 2.6360296,
    timeSimulation = 0.2357653,
    timeTotal = 2.9887292
end SimulationResult;
```

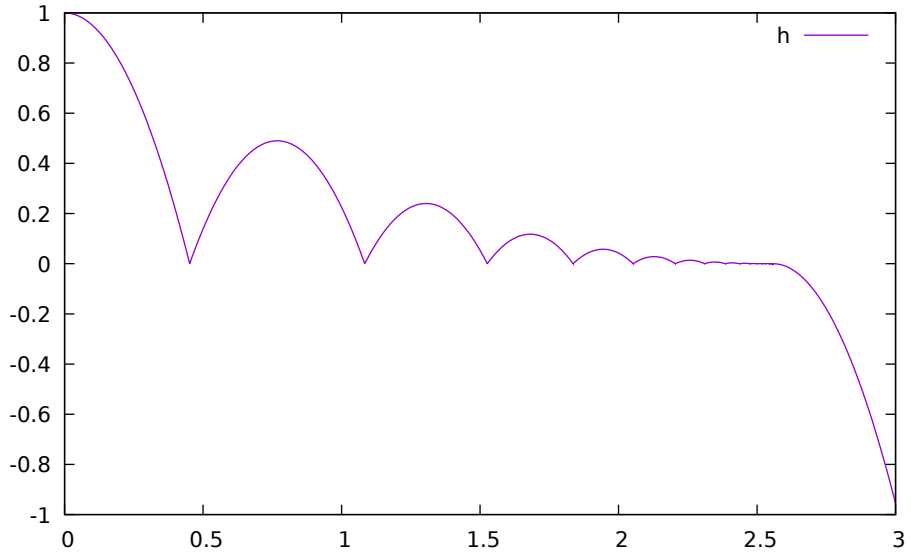


Figure 7.1: Height of the bouncing ball, simulated through an FMU.

7.2 Transmission Line Modeling (TLM) Based Co-Simulation

This chapter gives a short description how to get started using the TLM-Based co-simulation accessible via OMEdit.

The TLM Based co-simulation provides the following general functionalities:

- Import and add External non-Modelica models such as **Matlab/SimuLink**, **Adams**, and **BEAST** models
- Import and add External Modelica models e.g. from tools such as **Dymola** or **Wolfram SystemModeler**, etc.
- Specify startup methods and interfaces of the external model
- Build the composite models by connecting the external models

- Set the co-simulation parameters in the composite model
- Simulate the composite models using TLM based co-simulation

7.3 Composite Model Editing of External Models

The graphical composite model editor is an extension and specialization of the OpenModelica connection editor OMEdit. A composite model is composed of several external sub-models including the interconnections between these sub-models. External models are models which need not be in Modelica, they can be FMUs, or models accessed by proxies for co-simulation and connected by TLM-connections. The standard way to store a composite model is in an XML format. The XML schema standard is accessible from tlmModelDescription.xsd. Currently composite models can only be used for TLM based co-simulation of external models.

7.3.1 Loading a Composite Model for Co-Simulation

To load the composite model, select **File > Open Composite Model(s)** from the menu and select composite-model.xml.

OMEdit loads the composite model and show it in the **Libraries Browser**. Double-clicking the composite model in the **Libraries Browser** will display the composite model as shown below in Figure 7.2.

7.3.2 Co-Simulating the Composite Model

There are two ways to start co-simulation:

- Click **TLM Co-Simulation setup button** () from the toolbar (requires a composite model to be active in ModelWidget)
- Right click the composite model in the **Libraries Browser** and choose **TLM Co-Simulation setup** from the popup menu (see Figure 7.3)

The TLM Co-Simulation setup appears as shown below in Figure 7.4.

Click **Simulate** from the Co-simulation setup to confirm the co-simulation. Figure 7.5 will appears in which you will be able to see the progress information of the running co-simulation.

The editor also provides the means of reading the log files generated by the simulation manager and monitor. When the simulation ends, click **Open Manager Log File** or **Open Monitor Log File** from the co-simulation progress bar to check the log files.

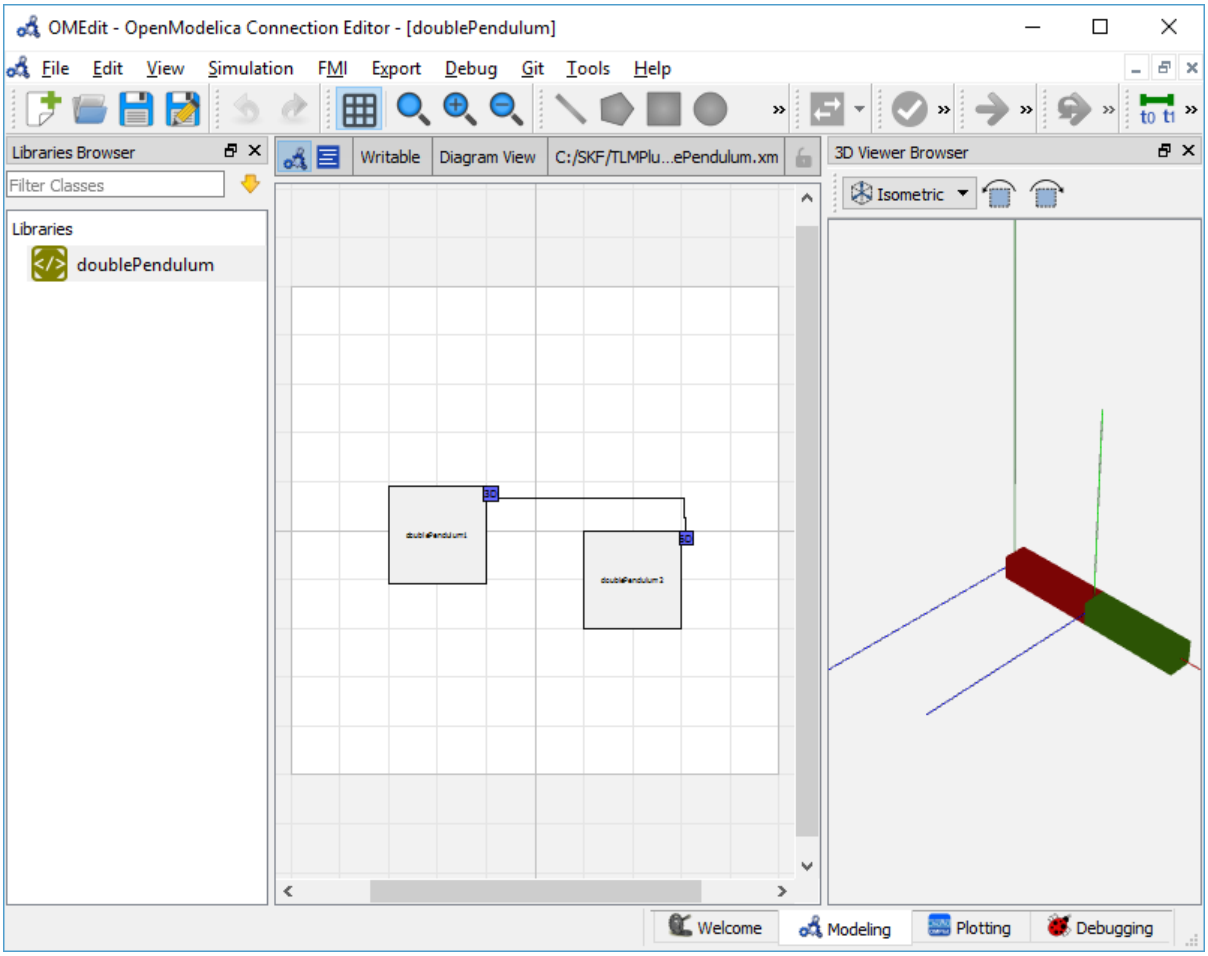


Figure 7.2: Composite Model with 3D View.

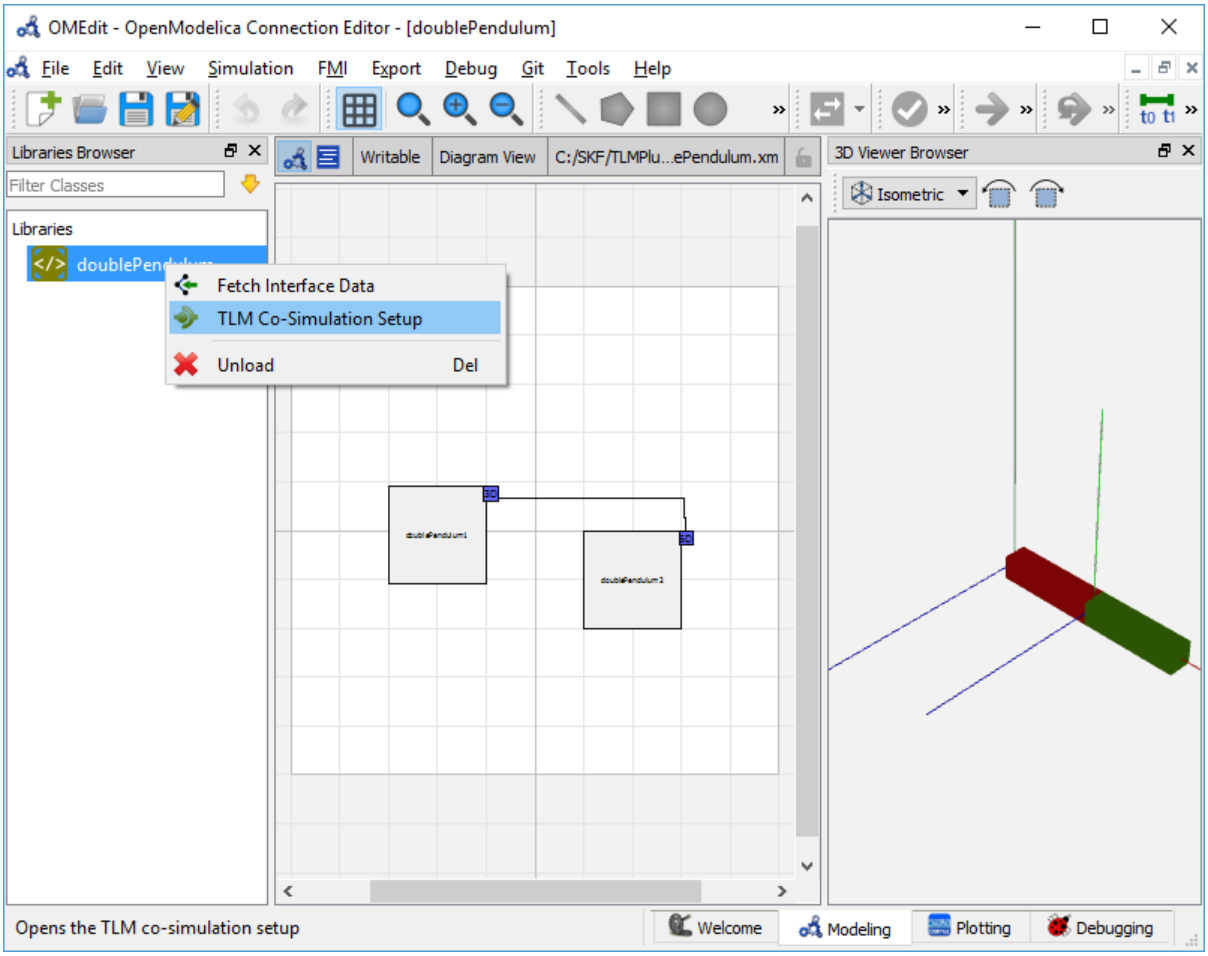


Figure 7.3: Co-simulating and Fetching Interface Data of a composite model from the Popup Menu .

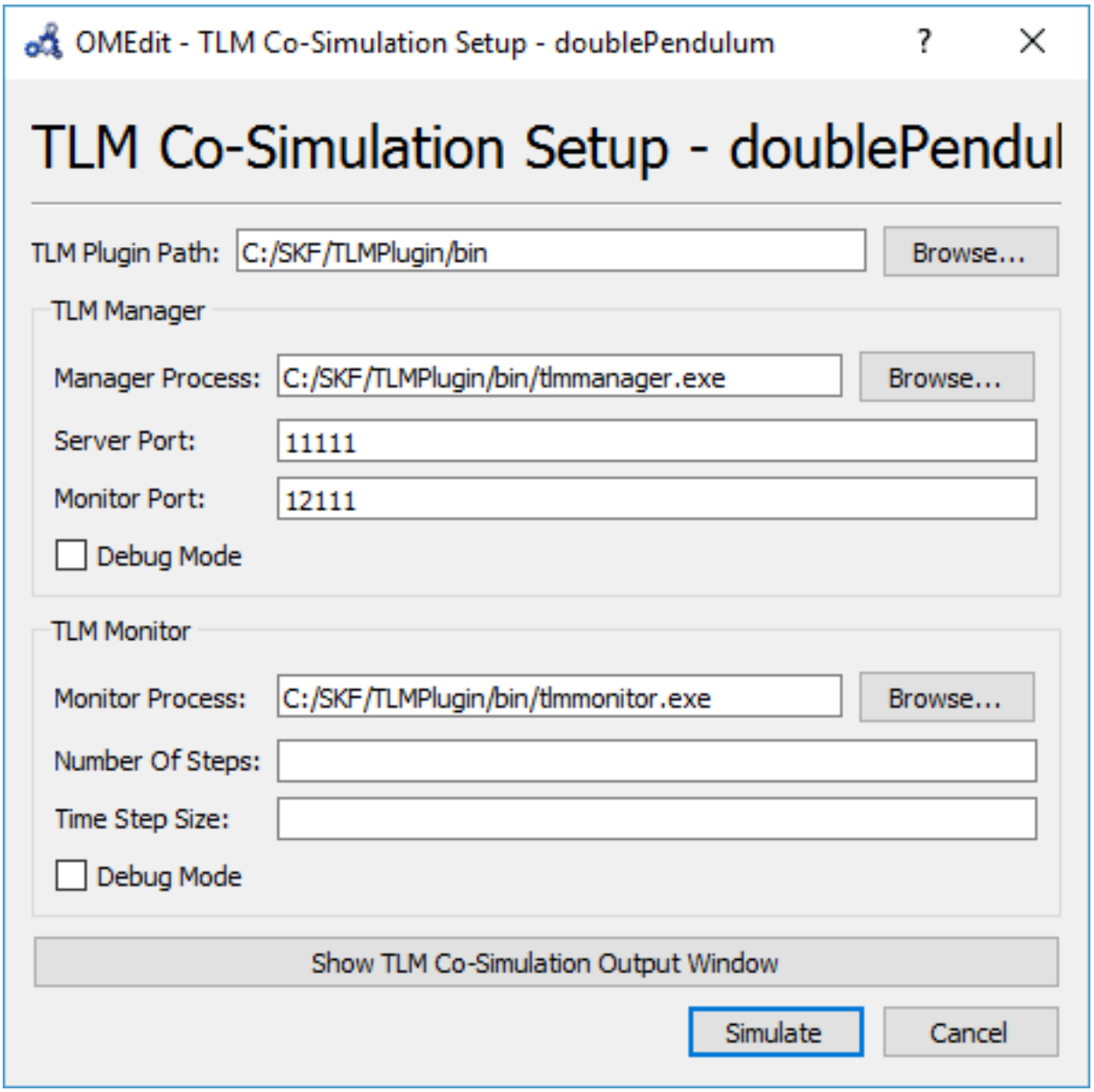


Figure 7.4: TLM Co-simulation Setup.

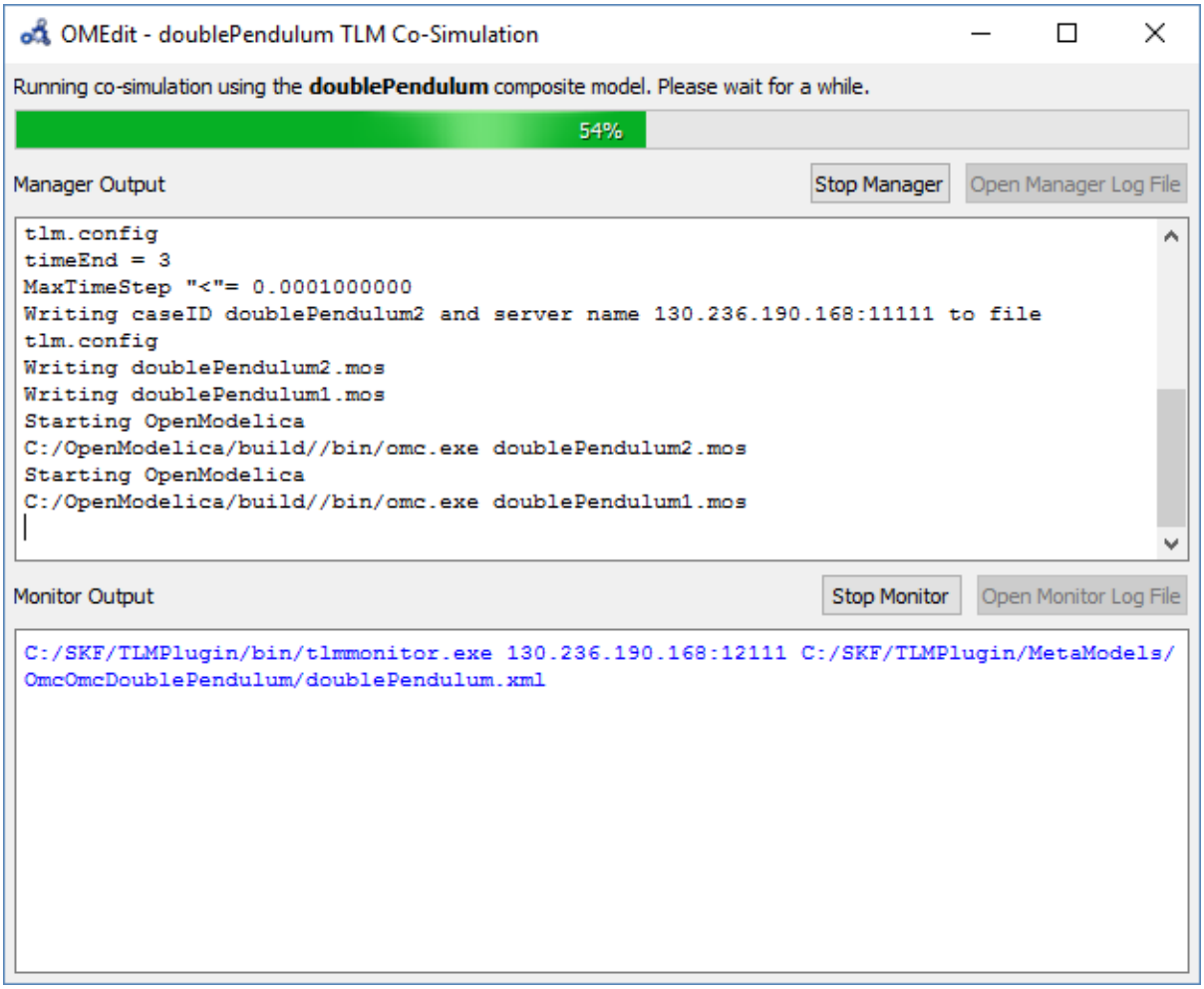


Figure 7.5: TLM Co-Simulation Progress.

7.3.3 Plotting the Simulation Results

When the co-simulation of the composite model is completed successful, simulation results are collected and visualized in the OMEdit plotting perspective as shown in Figure 7.6 and Figure 7.7. The **Variables Browser** display variables that can be plotted. Each variable has a checkbox, checking it will plot the variable.

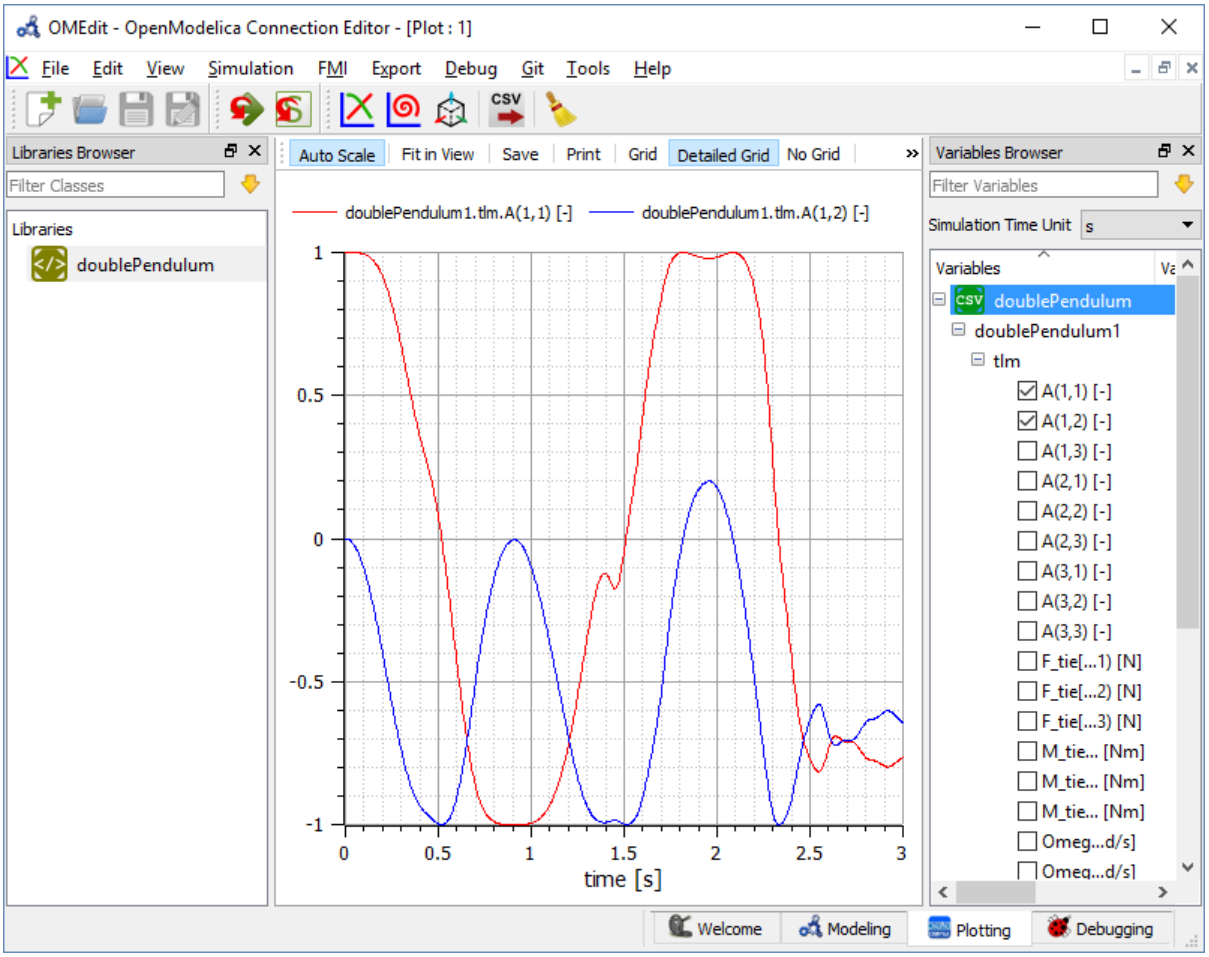


Figure 7.6: TLM Co-Simulation Results Plotting.

7.3.4 Preparing External Models

First step in co-simulation Modeling is to prepare the different external simulation models with TLM interfaces. Each external model belongs to a specific simulation tool, such as **MATLAB/Simulink***, **BEAST**, **MSC/ADAMS**, **Dymola** and **Wolfram SystemModeler**.

When the external models have all been prepared, the next step is to load external models in OMEdit by selecting the **File > Load External Model(s)** from the menu.

OMEdit loads the external model and show it in the **Libraries Browser** as shown below in Figure 7.8.

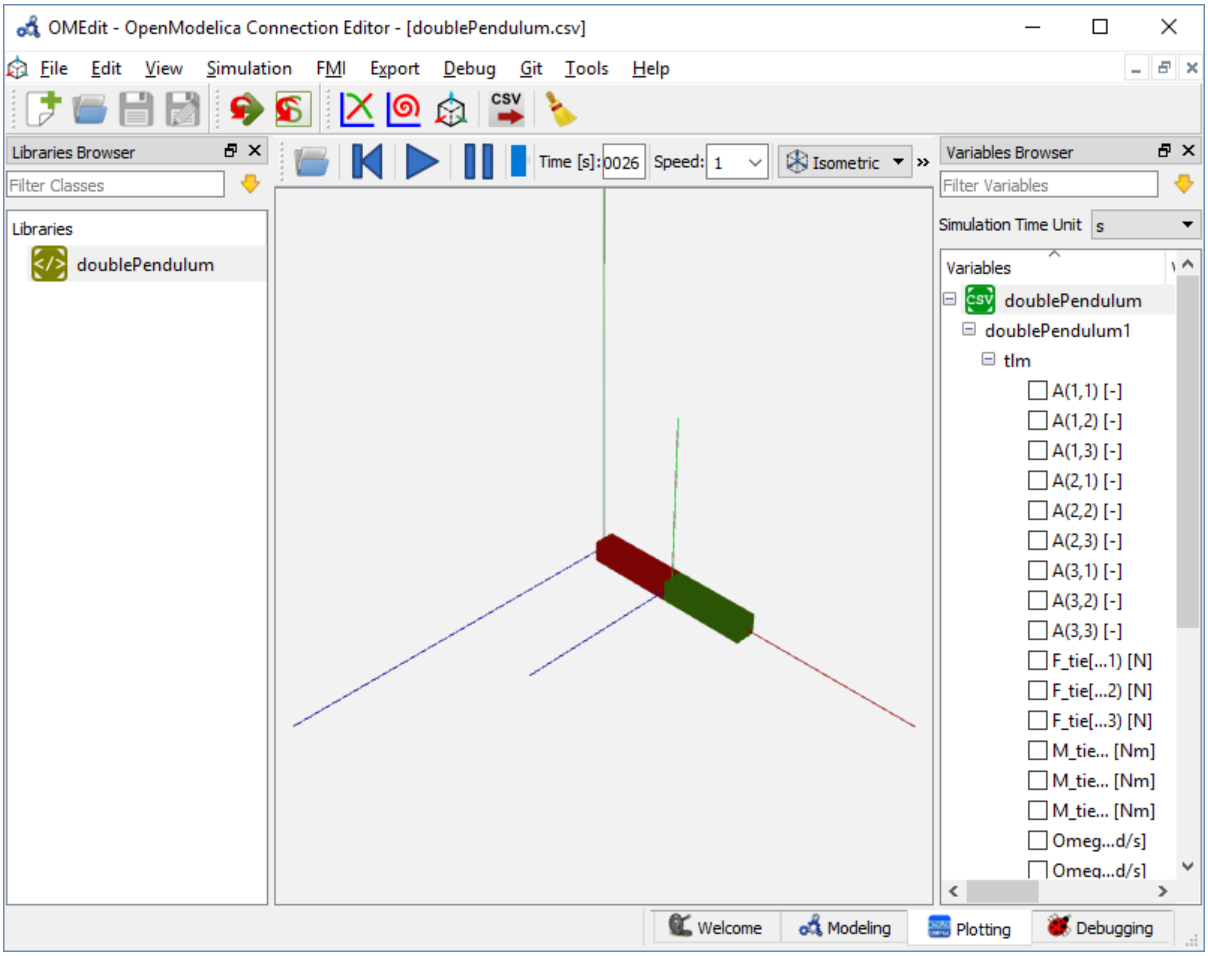


Figure 7.7: TLM Co-Simulation Visualization.

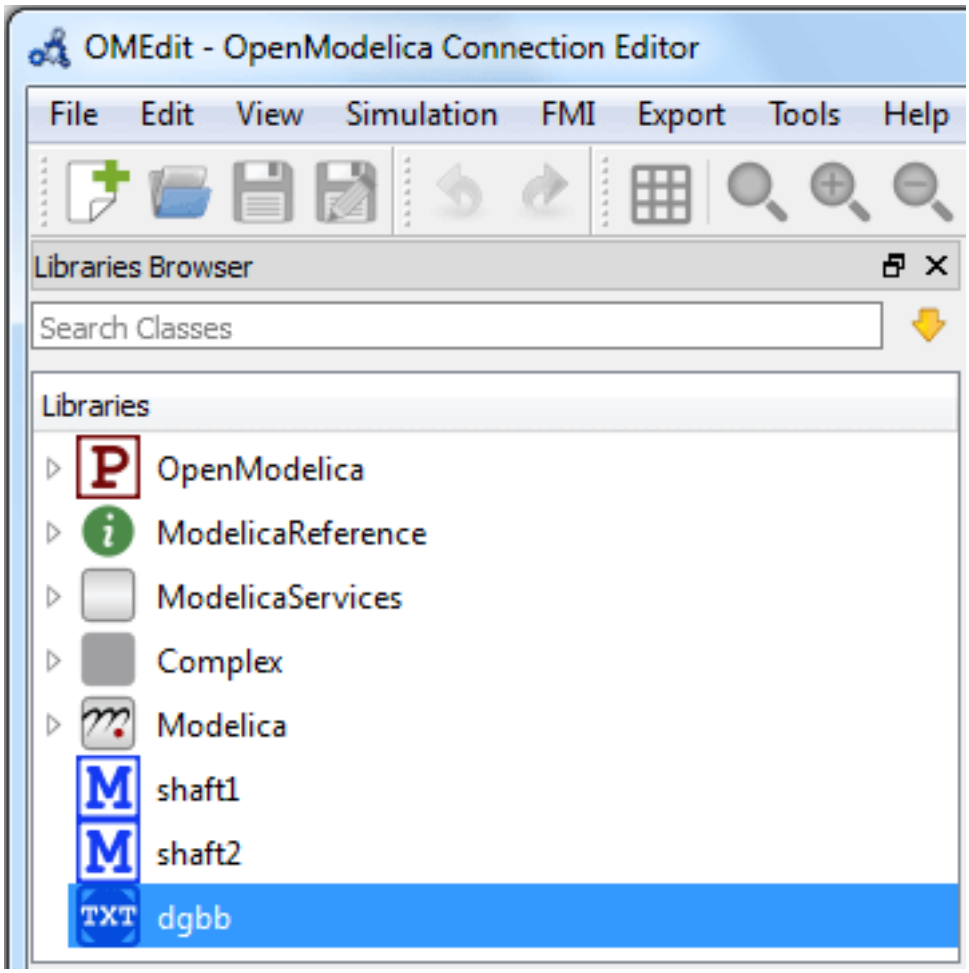


Figure 7.8: External Models in OMEdit.

7.3.5 Creating a New Composite Model

We will use the "Double pendulum" composite model which is a multibody system that consists of three sub-models: Two OpenModelica **Shaft** sub-models (**Shaft1** and **Shaft2**) and one **SKF/BEAST bearing** sub-model that together build a double pendulum. The **SKF/BEAST bearing** sub-model is a simplified model with only three balls to speed up the simulation. **Shaft1** is connected with a spherical joint to the world coordinate system. The end of **Shaft1** is connected via a TLM interface to the outer ring of the BEAST bearing model. The inner ring of the bearing model is connected via another TLM interface to **Shaft2**. Together they build the double pendulum with two **shafts**, one spherical OpenModelica joint, and one BEAST bearing.

To create a new composite model select **File > New Composite Model** from the menu.

Your new composite model will appear in the in the **Libraries Browser** once created. To facilitate the process of textual composite modeling and to provide users with a starting point, the **Text View** (see Figure 7.9) includes the composite model XML elements and the default simulation parameters.

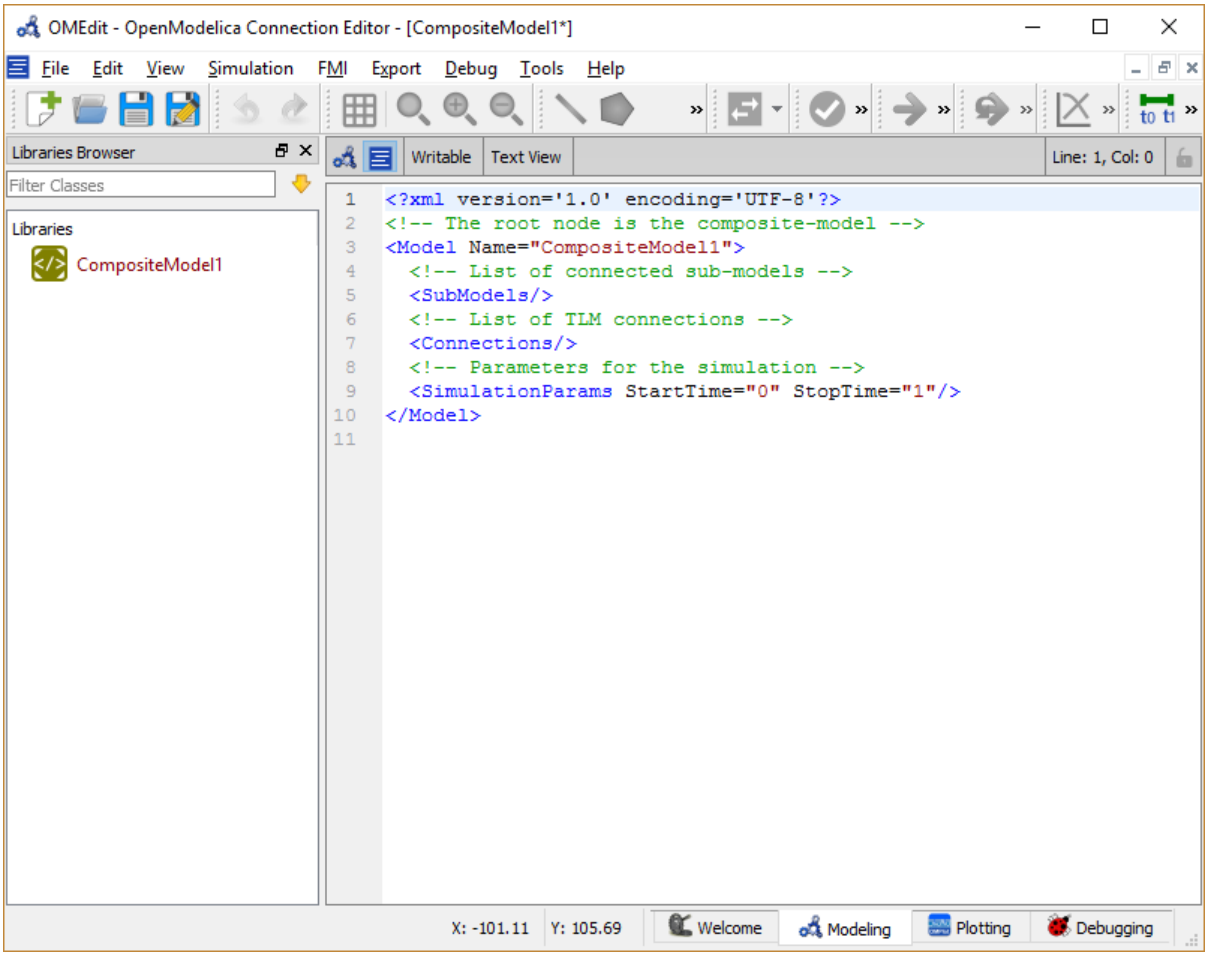


Figure 7.9: New composite model text view.

7.3.6 Adding Submodels

It is possible to build the double pendulum by drag-and-drop of each simulation model component (sub-model) from the **Libraries Browser** to the Diagram View. To place a component in the Diagram View of the double pendulum model, drag each external sub-model of the double pendulum (i.e. **Shaft1**, **Shaft2**, and **BEAST bearing** sub-model) from the **Libraries Browser** to the **Diagram View**.

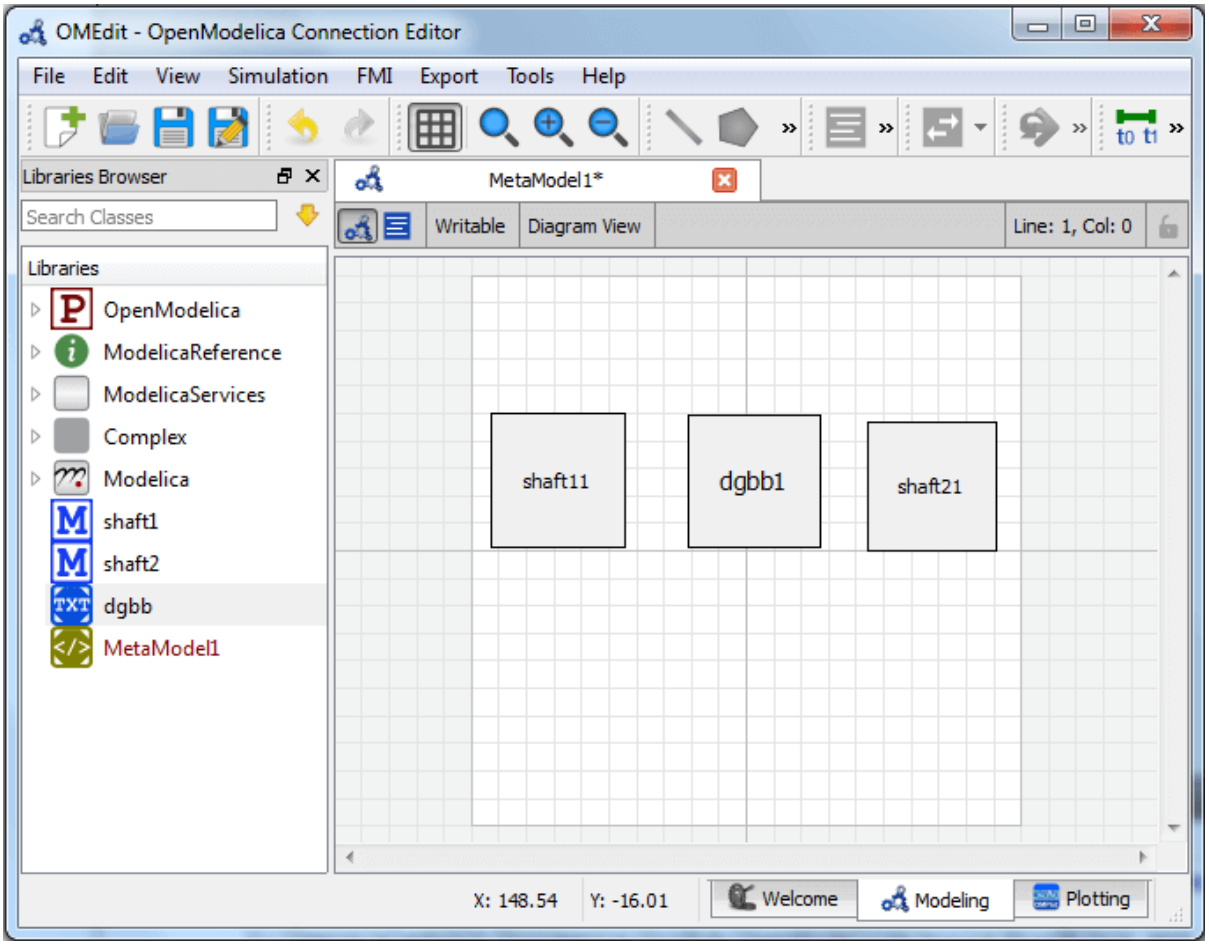


Figure 7.10: Adding sub-models to the double pendulum composite model.

7.3.7 Fetching Submodels Interface Data

To retrieve list of TLM interface data for sub-models, do any of the following methods:

- Click **Fetch Interface Data button** ( ) from the toolbar (requires a composite model to be active in ModelWidget)
- Right click the composite model in the **Library Browser** and choose **Fetch Interface Data** from the popup menu (see Figure 7.3).

To retrieve list of TLM interface data for a specific sub-model,

- Right click the sub-model inside the composite model and choose **Fetch Interface Data** from the popup menu.

Figure 7.11 will appear in which you will be able to see the progress information of fetching the interface data.

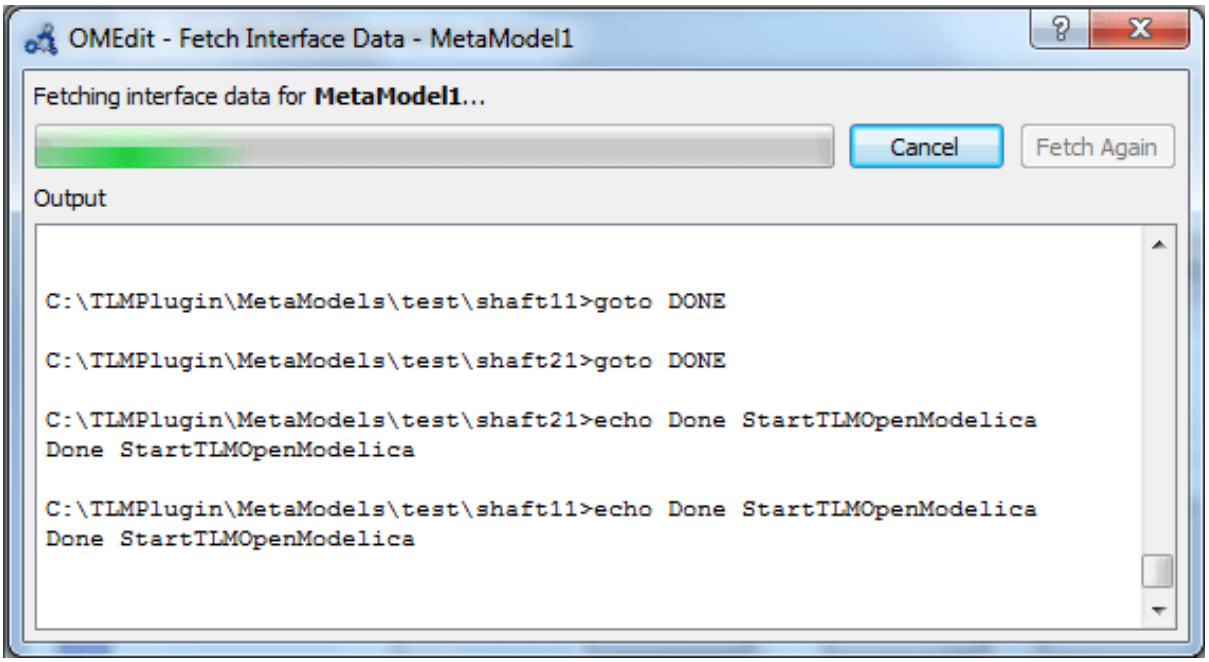


Figure 7.11: Fetching Interface Data Progress.

Once the TLM interface data of the sub-models are retrieved, the interface points will appear in the diagram view as shown below in Figure 7.12.

7.3.8 Connecting Submodels

When the sub-models and interface points have all been placed in the Diagram View, similar to Figure 7.12, the next step is to connect the sub-models. Sub-models are connected using the **Connection Line Button** ( ) from the toolbar.

To connect two sub-models, select the Connection Line Button and place the mouse cursor over an interface and click the left mouse button, then drag the cursor to the other sub-model interface, and click the left mouse button again. A connection dialog box as shown below in Figure 7.13 will appear in which you will be able to specify the connection attributes.

Continue to connect all sub-models until the composite model **Diagram View** looks like the one in Figure 7.14 below.

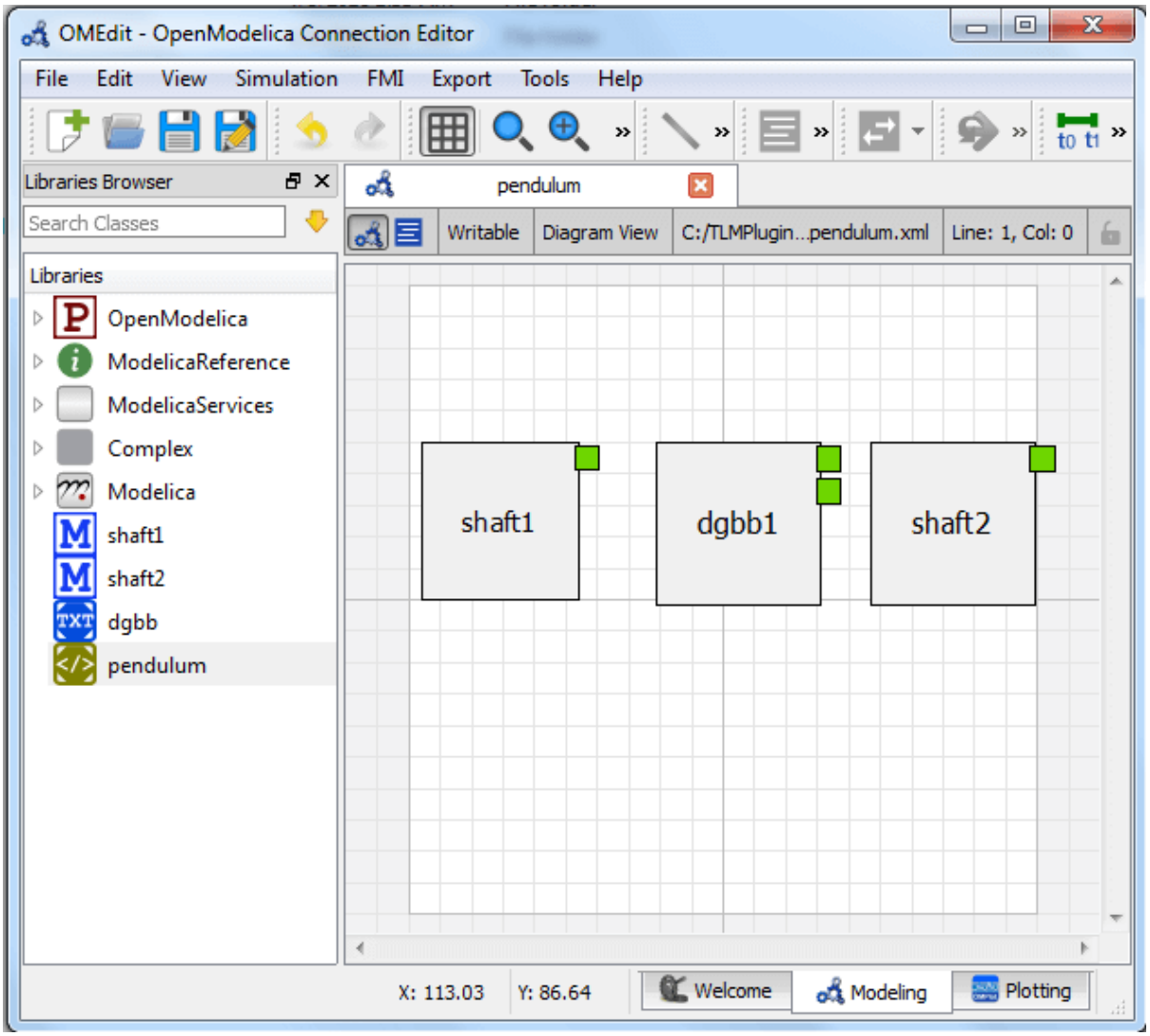


Figure 7.12: Fetching Interface Data.

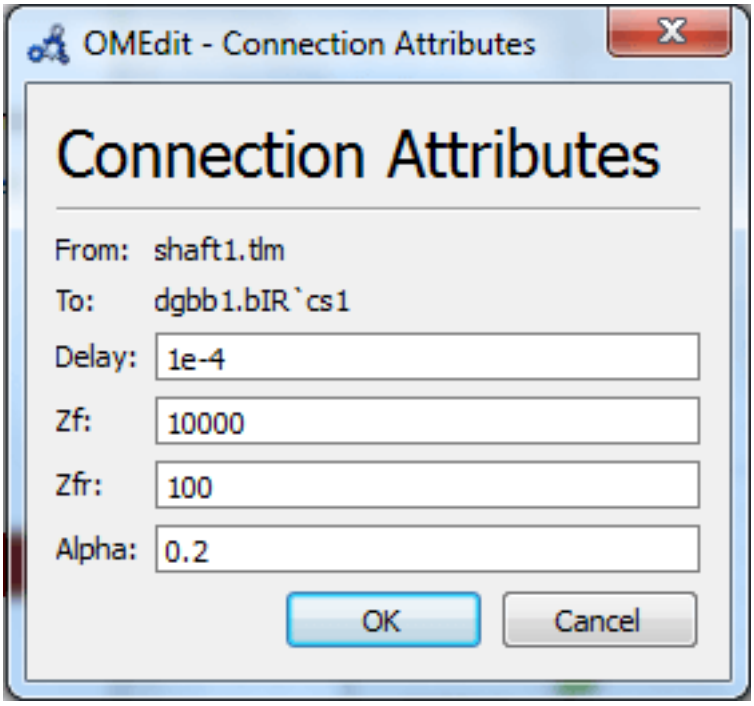


Figure 7.13: Sub-models Connection Dialog.

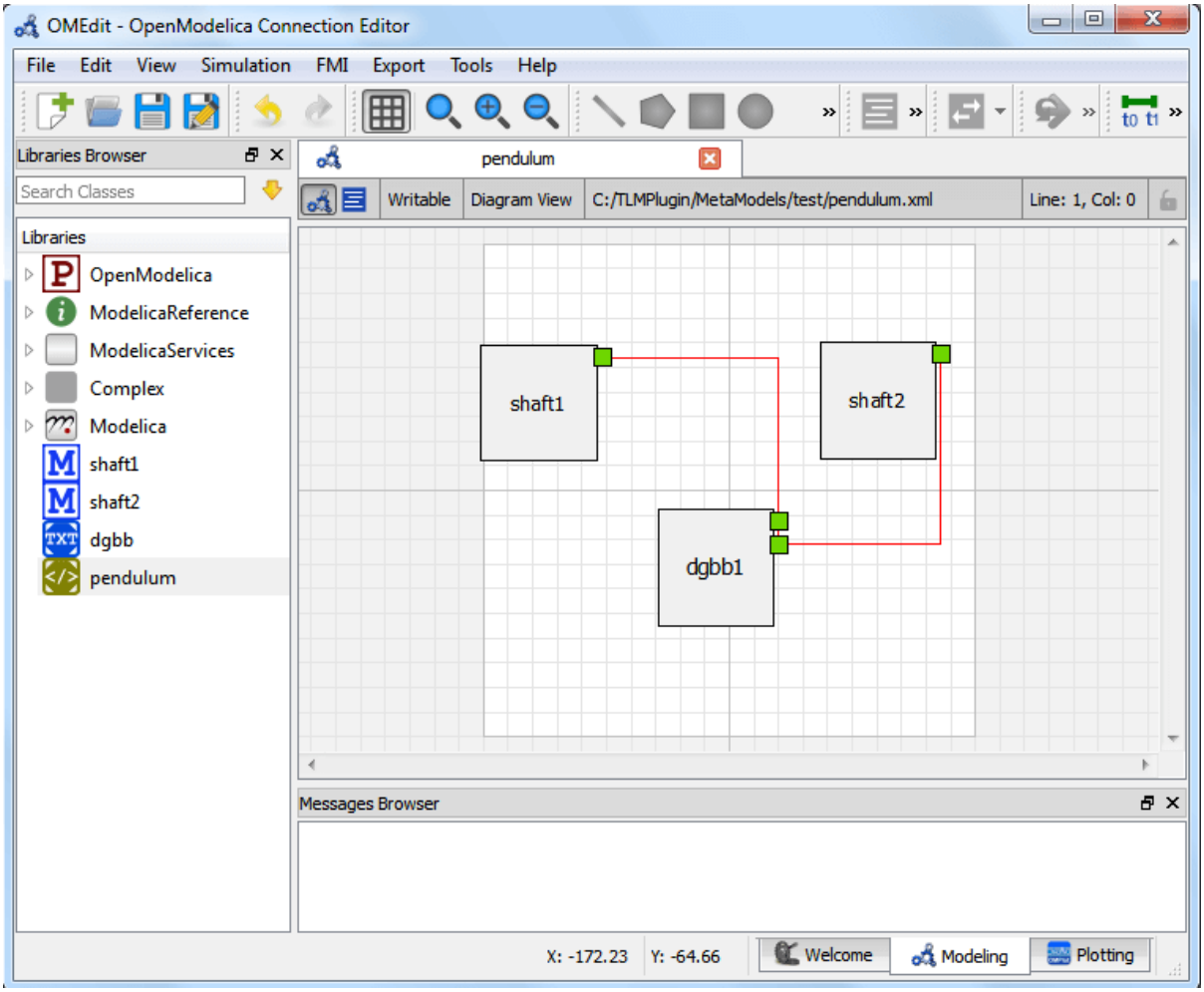


Figure 7.14: Connecting sub-models of the Double Pendulum Composite Model.

7.3.9 Changing Parameter Values of Submodels

To change a parameter value of a sub-model, do any of the following methods:

- Double-click on the sub-model you want to change its parameter
- Right click on the sub-model and choose **Attributes** from the popup menu

The parameter dialog of that sub-model appears as shown below in Figure 7.15 in which you will be able to specify the sub-models attributes.

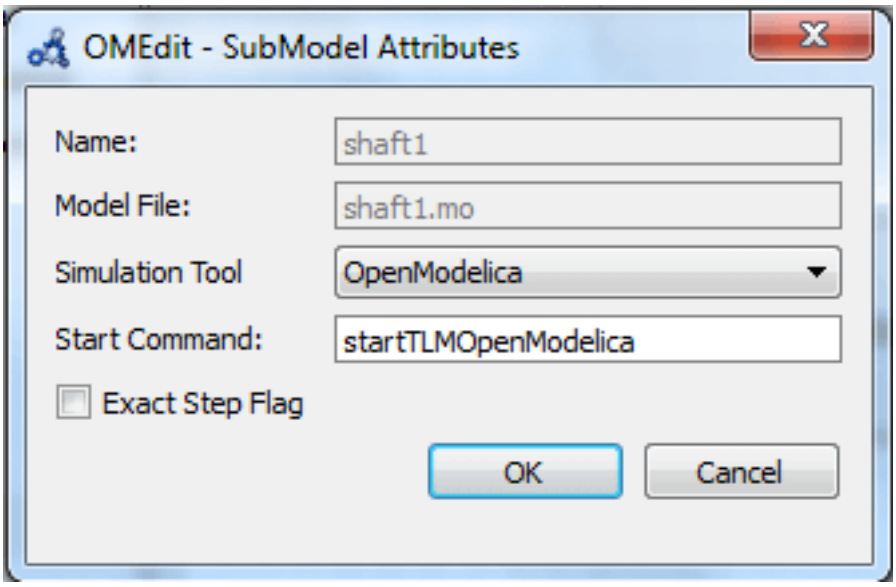


Figure 7.15: Changing Parameter Values of Sub-models Dialog.

7.3.10 Changing Parameter Values of Connections

To change a parameter value of a connection, do any of the following methods:

- Double-click on the connection you want to change its parameter
- Right click on the connection and choose **Attributes** from the popup menu.

The parameter dialog of that connection appears (see Figure 7.13) in which you will be able to specify the connections attributes.

7.3.11 Changing Co-Simulation Parameters

To change the co-simulation parameters, do any of the following methods:

- Click Simulation Parameters button ( ) from the toolbar (requires a composite model to be active in ModelWidget)
- Right click an empty location in the Diagram View of the composite model and choose **Simulation Parameters** from the popup menu (see Figure 7.16)

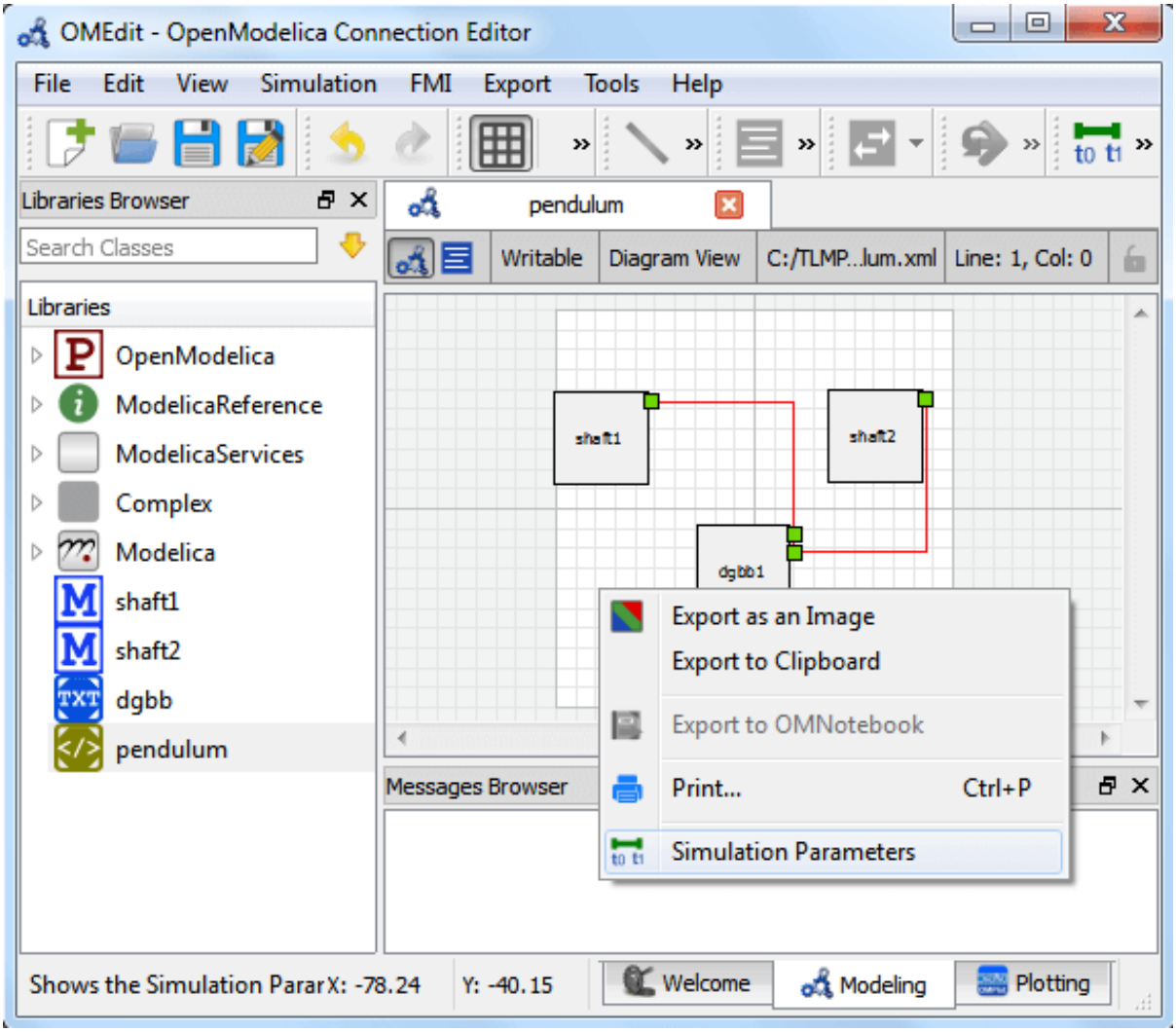


Figure 7.16: Changing Co-Simulation Parameters from the Popup Menu.

The co-simulation parameter dialog of the composite model appears as shown below in Figure 7.17 in which you will be able to specify the simulation parameters.

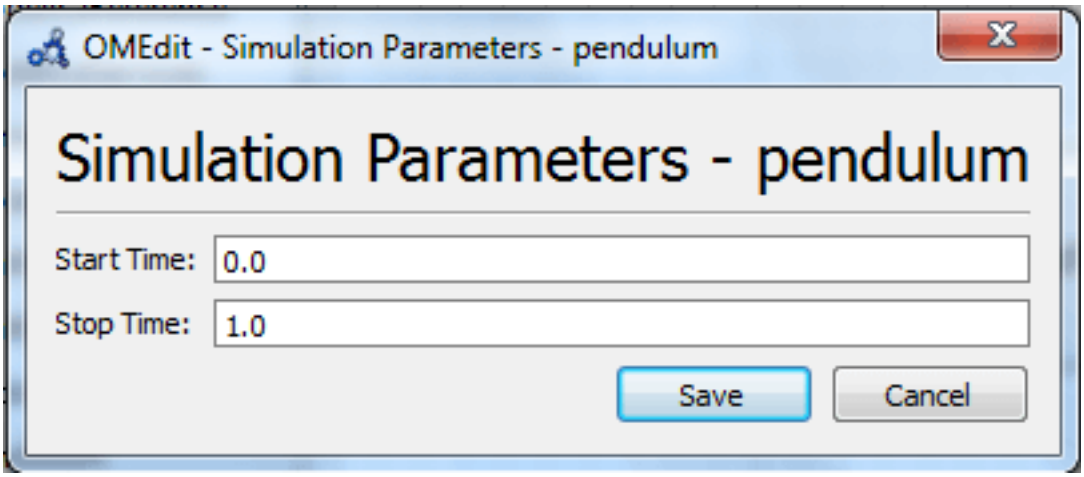


Figure 7.17: Changing Co-Simulation Parameters Dialog.

第8章　OMSimulator

OMSimulator has its own documentation.

第9章　OpenModelica Encryption

The encryption module allows the library developers to encrypt their libraries for different platforms. Note that you need a special version of OpenModelica with encryption support. Contact us if you want one.

9.1 Encrypting the Library

In order to encrypt the Modelica package call *buildEncryptedPackage(TopLevelPackageName)* from mos script or from **OMEdit** right click the package in Libraries Browser and select *Export Encrypted Package* or select *Export > Export Encrypted Package* from the menu.

All the Modelica files are encrypted and the whole library is zipped into a single file i.e., *PackageName.mol*. Note that you can only encrypt Modelica packages saved in a folder structure. The complete folder structure remains as it is. No encryption is done on the resource files.

9.2 Loading an Encrypted Library

To load the encrypted package call *loadEncryptedPackage(EncryptedPackage.mol)* from the mos script or from **OMEdit** *File > Load Encrypted Package*.

9.3 Notes

- There is no license management and obfuscation of the generated code and files. However just a basic encryption and decryption is supported along with full support for protection access annotation as defined in Modelica specification 18.9. This means that anyone who has an OpenModelica version with encryption support can encrypt or decrypt files.
- OpenModelica encryption is based on SEMLA (Safe/Superiour/Super Encryption of Modelica Libraries and Artifacts) module from Modelon AB.

第10章 OMNotebook with DrModelica and DrControl

This chapter covers the OpenModelica electronic notebook subsystem, called OMNotebook, together with the DrModelica tutoring system for teaching Modelica, and DrControl for teaching control together with Modelica. Both are using such notebooks.

10.1 Interactive Notebooks with Literate Programming

Interactive Electronic Notebooks are active documents that may contain technical computations and text, as well as graphics. Hence, these documents are suitable to be used for teaching and experimentation, simulation scripting, model documentation and storage, etc.

10.1.1 Mathematica Notebooks

Literate Programming [Knu84] is a form of programming where programs are integrated with documentation in the same document. Mathematica notebooks [Wol96] is one of the first WYSIWYG (What-You-See-Is-What-You-Get) systems that support Literate Programming. Such notebooks are used, e.g., in the MathModelica modeling and simulation environment, see e.g. Figure 10.1 below and Chapter 19 in [Fri04].

10.1.2 OMNotebook

The OMNotebook software [Axe05][Fernstrom06] is a new open source free software that gives an interactive WYSIWYG realization of Literate Programming, a form of programming where programs are integrated with documentation in the same document.

The OMNotebook facility is actually an interactive WYSIWYG realization of Literate Programming, a form of programming where programs are integrated with documentation in the same document. OMNotebook is a simple open-source software tool for an electronic notebook supporting Modelica.

A more advanced electronic notebook tool, also supporting mathematical typesetting and many other facilities, is provided by Mathematica notebooks in the MathModelica environment, see Figure 10.1.

Traditional documents, e.g. books and reports, essentially always have a hierarchical structure. They are divided into sections, subsections, paragraphs, etc. Both the document itself and its sections usually have headings as labels for easier navigation. This kind of structure is also reflected in electronic notebooks. Every notebook corresponds to one document (one file) and contains a tree structure of cells. A cell can have different kinds of

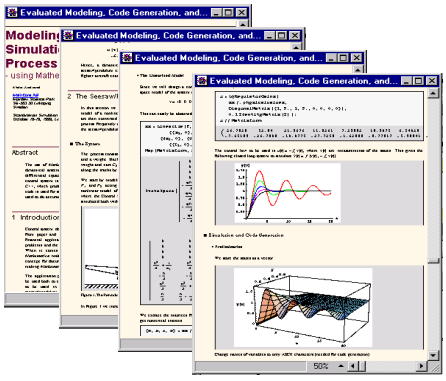


Figure 10.1: Examples of Mathematica notebooks in the MathModelica modeling and simulation environment.

contents, and can even contain other cells. The notebook hierarchy of cells thus reflects the hierarchy of sections and subsections in a traditional document such as a book.

10.2 DrModelica Tutoring System – an Application of OMNotebook

Understanding programs is hard, especially code written by someone else. For educational purposes it is essential to be able to show the source code and to give an explanation of it at the same time.

Moreover, it is important to show the result of the source code's execution. In modeling and simulation it is also important to have the source code, the documentation about the source code, the execution results of the simulation model, and the documentation of the simulation results in the same document. The reason is that the problem solving process in computational simulation is an iterative process that often requires a modification of the original mathematical model and its software implementation after the interpretation and validation of the computed results corresponding to an initial model.

Most of the environments associated with equation-based modeling languages focus more on providing efficient numerical algorithms rather than giving attention to the aspects that should facilitate the learning and teaching of the language. There is a need for an environment facilitating the learning and understanding of Modelica. These are the reasons for developing the DrModelica teaching material for Modelica and for teaching modeling and simulation.

An earlier version of DrModelica was developed using the MathModelica (now Wolfram SystemModeler) environment. The rest of this chapter is concerned with the OMNotebook version of DrModelica and on the OMNotebook tool itself.

DrModelica has a hierarchical structure represented as notebooks. The front-page notebook is similar to a table of contents that holds all other notebooks together by providing links to them. This particular notebook is the first page the user will see (Figure 10.2).

In each chapter of DrModelica the user is presented a short summary of the corresponding chapter of the Modelica book [Fri04]. The summary introduces some *keywords*, being hyperlinks that will lead the user to other notebooks describing the keywords in detail.

Now, let us consider that the link "*HelloWorld*" in DrModelica Section is clicked by the user. The new HelloWorld notebook (see Figure 10.3), to which the user is being linked, is not only a textual description but also contains one or more examples explaining the specific keyword. In this class, HelloWorld, a differential equation is specified.

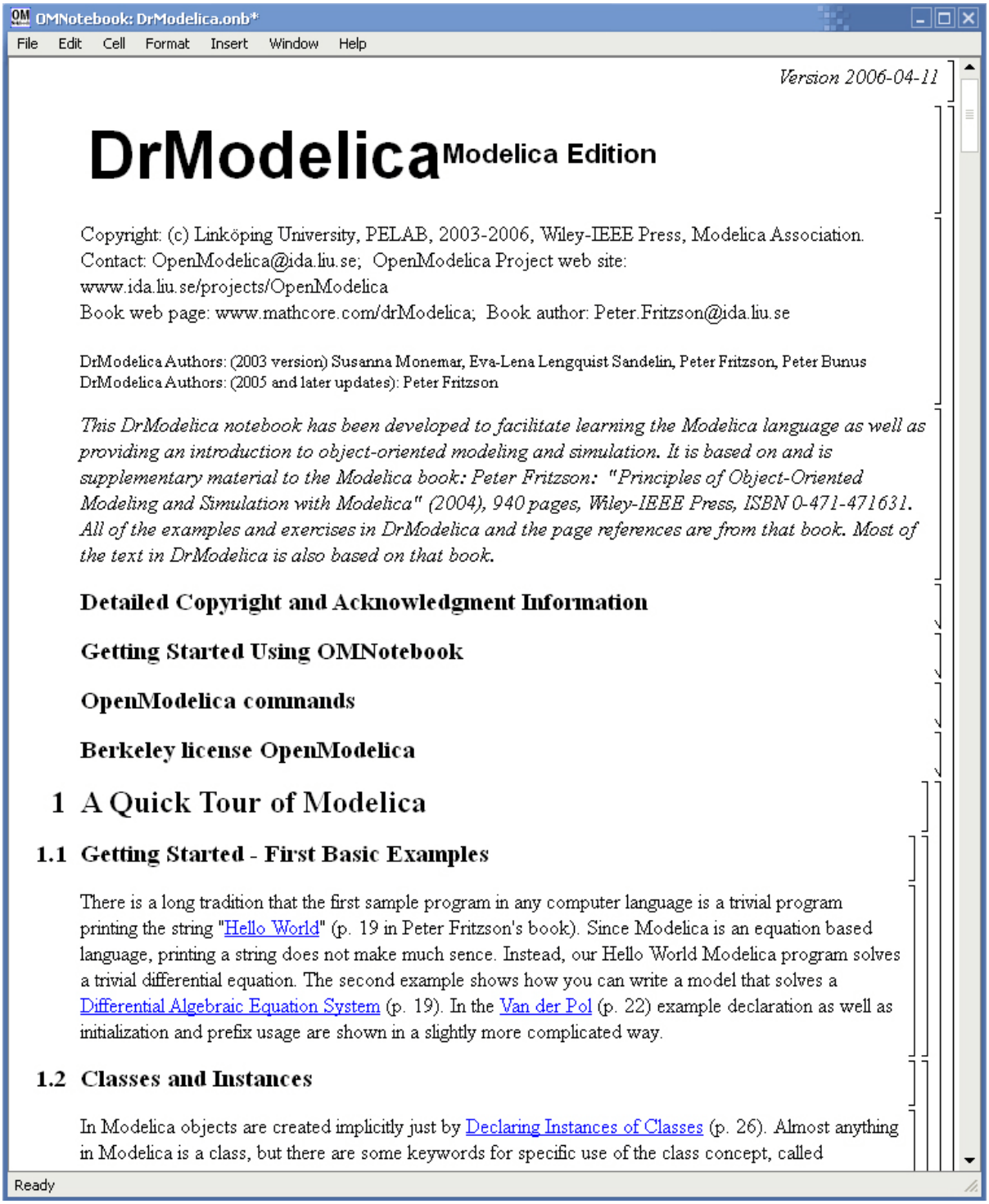


Figure 10.2: The front-page notebook of the OMNotebook version of the DrModelica tutoring system.

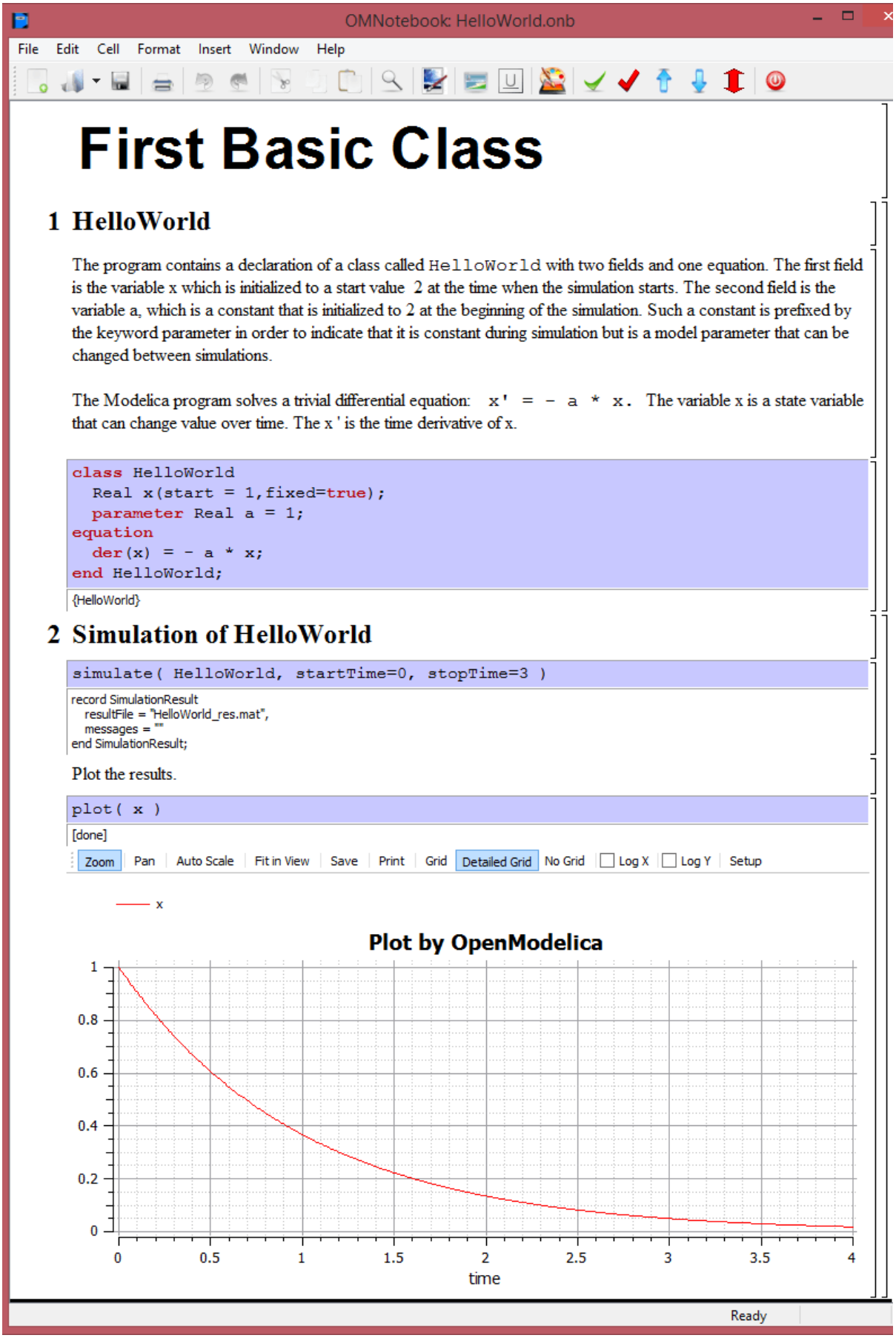


Figure 10.3: The HelloWorld class simulated and plotted using the OMNotebook version of DrModelica.

No information in a notebook is fixed, which implies that the user can add, change, or remove anything in a notebook. Alternatively, the user can create an entirely new notebook in order to write his/her own programs or copy examples from other notebooks. This new notebook can be linked from existing notebooks.

When a class has been successfully evaluated the user can simulate and plot the result, as previously depicted in Figure 10.3 for the simple HelloWorld example model.

After reading a chapter in DrModelica the user can immediately practice the newly acquired information by doing the exercises that concern the specific chapter. Exercises have been written in order to elucidate language constructs step by step based on the pedagogical assumption that a student learns better "*using the strategy of learning by doing*". The exercises consist of either theoretical questions or practical programming assignments. All exercises provide answers in order to give the user immediate feedback.

Figure 10.4 shows part of Chapter 9 of the DrModelica teaching material. Here the user can read about language constructs, like algorithm sections, when-statements, and reinit equations, and then practice these constructs by solving the exercises corresponding to the recently studied section.

Exercise 1 from Chapter 9 is shown in Figure 10.5. In this exercise the user has the opportunity to practice different language constructs and then compare the solution to the answer for the exercise. Notice that the answer is not visible until the *Answer* section is expanded. The answer is shown in Figure 10.6.

10.3 DrControl Tutorial for Teaching Control Theory

DrControl is an interactive OMNotebook document aimed at teaching control theory. It is included in the OpenModelica distribution and appears under the directory:

```
>>> getInstallationDirectoryPath() + "/share/omnotebook/drcontrol"
"<<OPENMODELICAHOME>>/share/omnotebook/drcontrol"
```

The front-page of DrControl resembles a linked table of content that can be used as a navigation center. The content list contains topics like:

- Getting started
- The control problem in ordinary life
- Feedback loop
- Mathematical modeling
- Transfer function
- Stability
- Example of controlling a DC-motor
- Feedforward compensation
- State-space form
- State observation
- Closed loop control system.

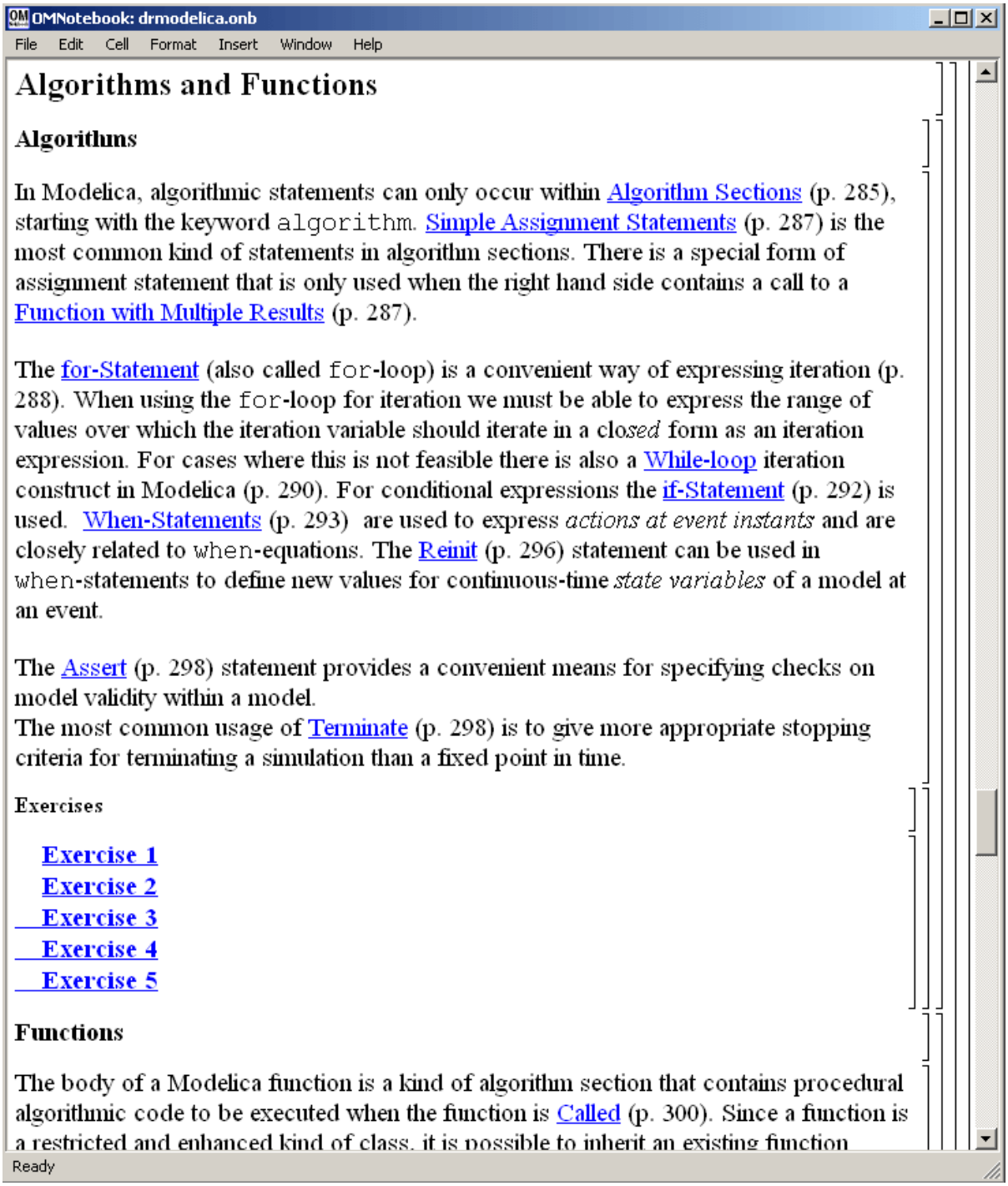


Figure 10.4: DrModelica Chapter on Algorithms and Functions in the main page of the OMNotebook version of DrModelica.

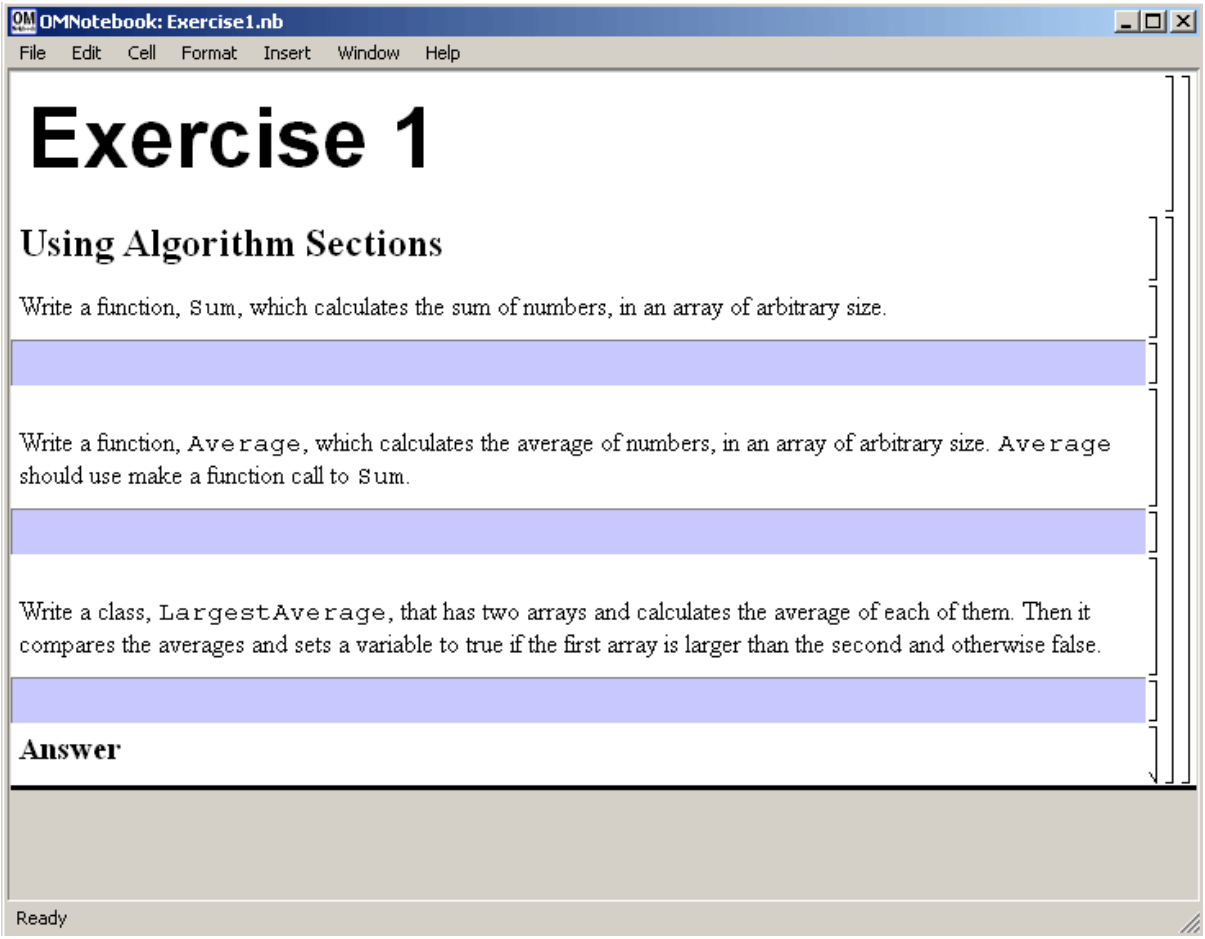


Figure 10.5: Exercise 1 in Chapter 9 of DrModelica.

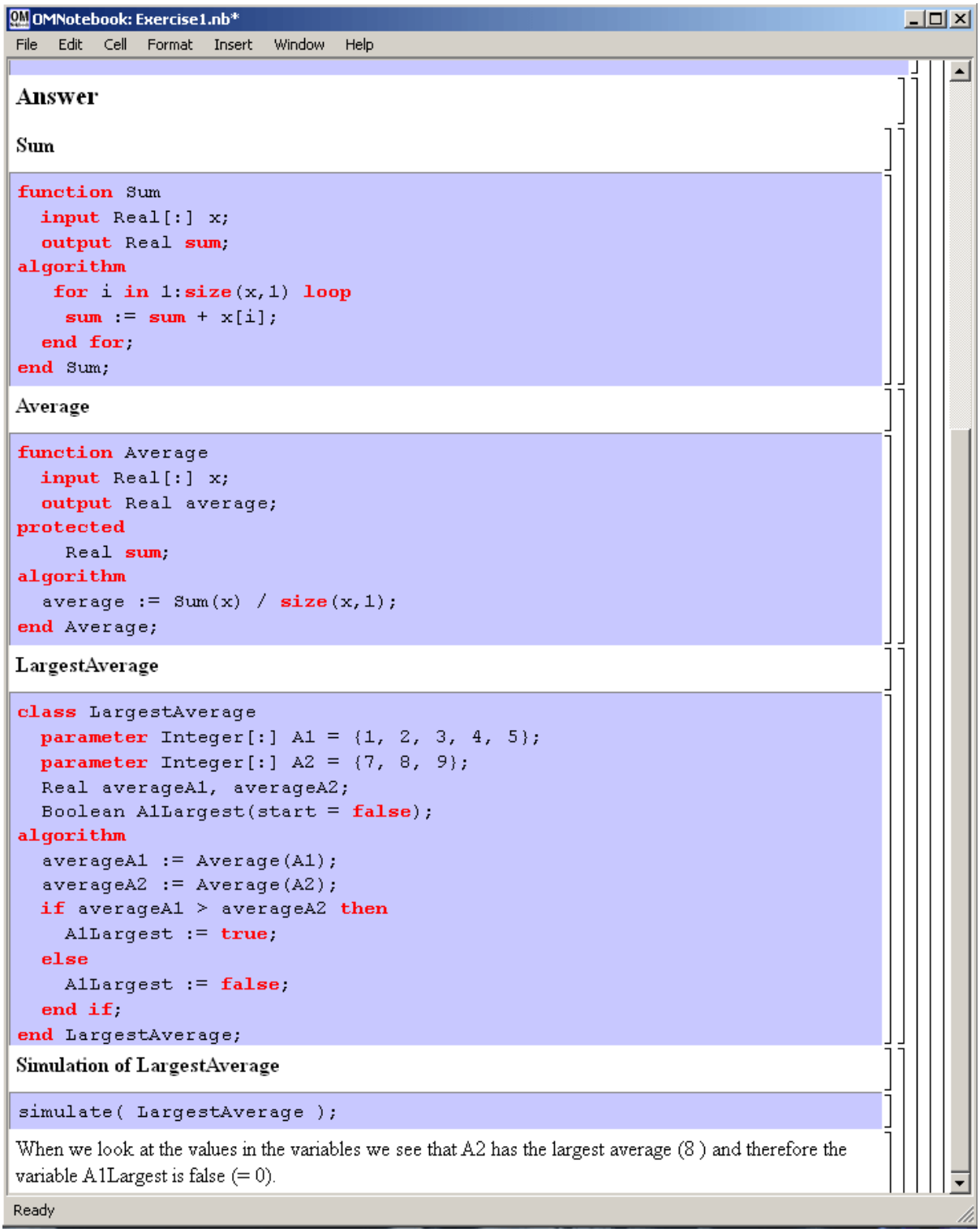


Figure 10.6: The answer section to Exercise 1 in Chapter 9 of DrModelica.

- Reconstructed system
- Linear quadratic optimization
- Linearization

Each entry in this list leads to a new notebook page where either the theory is explained with Modelica examples or an exercise with a solution is provided to illustrate the background theory. Below we show a few sections of DrControl.

10.3.1 Feedback Loop

One of the basic concepts of control theory is using feedback loops either for neutralizing the disturbances from the surroundings or a desire for a smoother output.

In Figure 10.7, control of a simple car model is illustrated where the car velocity on a road is controlled, first with an open loop control, and then compared to a closed loop system with a feedback loop. The car has a mass m, velocity y, and aerodynamic coefficient α. The θ is the road slope, which in this case can be regarded as noise.

Lets look at the Modelica model for the open loop controlled car:

$$m\dot{y} = u - \alpha y - mg * sin(\theta)$$

```
model noFeedback
  import SI = Modelica.SIunits;
  SI.Velocity y;                                  // output signal without noise,␣
↪theta = 0 -> v(t) = 0
  SI.Velocity yNoise;                             // output signal with noise,    ␣
↪theta <> 0 -> v(t) <> 0
  parameter SI.Mass m = 1500;
  parameter Real alpha = 200;
  parameter SI.Angle theta = 5*3.141592/180;
  parameter SI.Acceleration g = 9.82;
  SI.Force u;
  SI.Velocity r=20;
equation
  m*der(y)=u-alpha*y;                              // signal without noise
  m*der(yNoise)=u-alpha*yNoise-m*g*sin(theta); // with noise
  u = 250*r;
end noFeedback;
```

By applying a road slope angle different from zero the car velocity is influenced which can be regarded as noise in this model. The output signal in Figure 10.8 is stable but an overshoot can be observed compared to the reference signal. Naturally the overshoot is not desired and the student will in the next exercise learn how to get rid of this undesired behavior of the system.

```
>>> loadModel(Modelica)
true
>>> simulate(noFeedback, stopTime=100)
record SimulationResult
    resultFile = "<<DOCHOME>>/noFeedback_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 100.0, numberOfIntervals =␣
↪500, tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'noFeedback', options␣
↪= '', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
```

(次のページに続く)

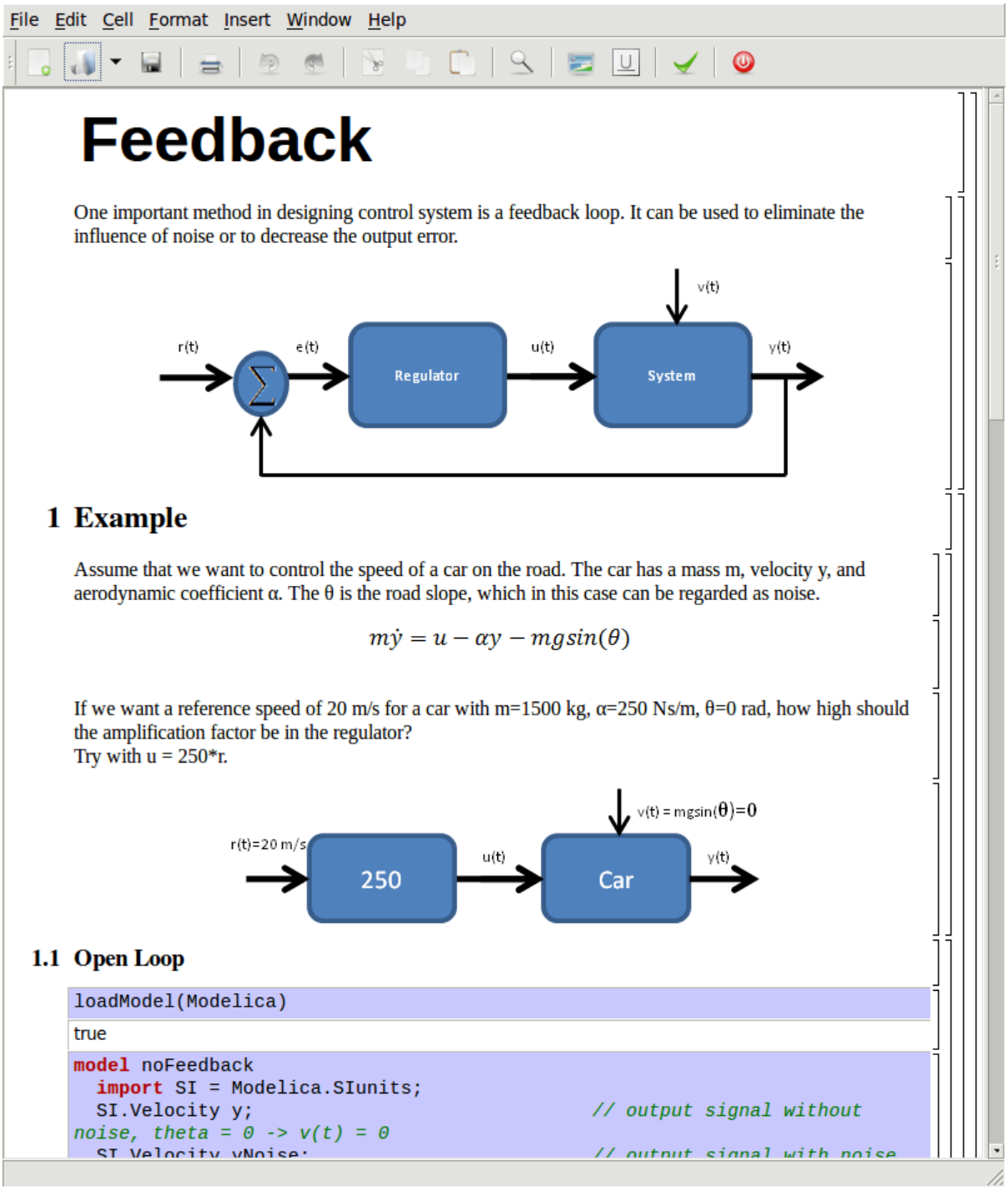


Figure 10.7: Feedback loop.

(前のページからの続き)

```
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.7245374,
    timeBackend = 0.0148939,
    timeSimCode = 0.005702100000000001,
    timeTtemplates = 0.0148072,
    timeCompile = 2.4926933,
    timeSimulation = 0.1105966,
    timeTotal = 3.3652542
end SimulationResult;
```

警告:

Warning: The initial conditions are not fully specified. For more information set -d=initialization. In OMEdit Tools->Options->Simulation->OMCFlags, in OMNotebook call setCommandLineOptions("-d=initialization").

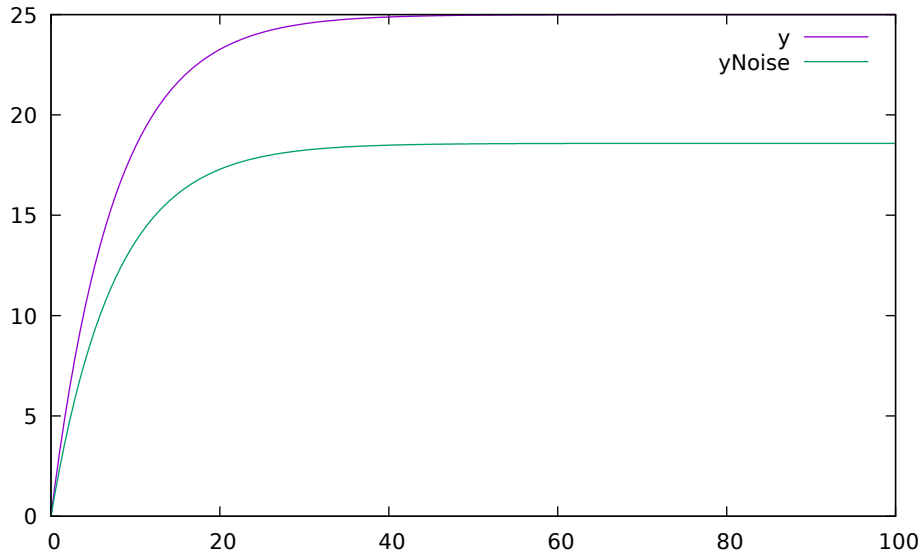


Figure 10.8: Open loop control example.

The closed car model with a proportional regulator is shown below:

$$u = K * (r - y)$$

```
model withFeedback
  import SI = Modelica.SIunits;
  SI.Velocity y;                                  // output signal with feedback␣
↪link and without noise, theta = 0 -> v(t) = 0
  SI.Velocity yNoise;                             // output signal with feedback␣
↪link and noise,    theta <> 0 -> v(t) <> 0
  parameter SI.Mass m = 1500;
  parameter Real alpha = 250;
  parameter SI.Angle theta = 5*3.141592/180;
  parameter SI.Acceleration g = 9.82;
  SI.Force u;
```

(次のページに続く)

(前のページからの続き)

```
  SI.Force uNoise;
  SI.Velocity r=20;
equation
  m*der(y)=u-alpha*y;
  m*der(yNoise)=uNoise-alpha*yNoise-m*g*sin(theta);
  u = 5000*(r-y);
  uNoise = 5000*(r-yNoise);
end withFeedback;
```

By using the information about the current level of the output signal and re-tune the regulator the output quantity can be controlled towards the reference signal smoothly and without an overshoot, as shown in Figure 10.9.

In the above simple example the flat modeling approach was adopted since it was the fastest one to quickly obtain a working model. However, one could use the object oriented approach and encapsulate the car and regulator models in separate classes with the Modelica connector mechanism in between.

```
>>> loadModel(Modelica)
true
>>> simulate(withFeedback, stopTime=10)
record SimulationResult
    resultFile = "<<DOCHOME>>/withFeedback_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 10.0, numberOfIntervals = 500,
 tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'withFeedback', options = '
↪', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.2761081999999999,
    timeBackend = 0.005802600000000001,
    timeSimCode = 0.0015777,
    timeTemplates = 0.0106669,
    timeCompile = 2.5875446,
    timeSimulation = 0.1103352,
    timeTotal = 2.9941973
end SimulationResult;
```

警告:

Warning: The initial conditions are not fully specified. For more information set -d=initialization. In OMEdit Tools->Options->Simulation->OMCFlags, in OMNotebook call setCommandLineOptions("-d=initialization").

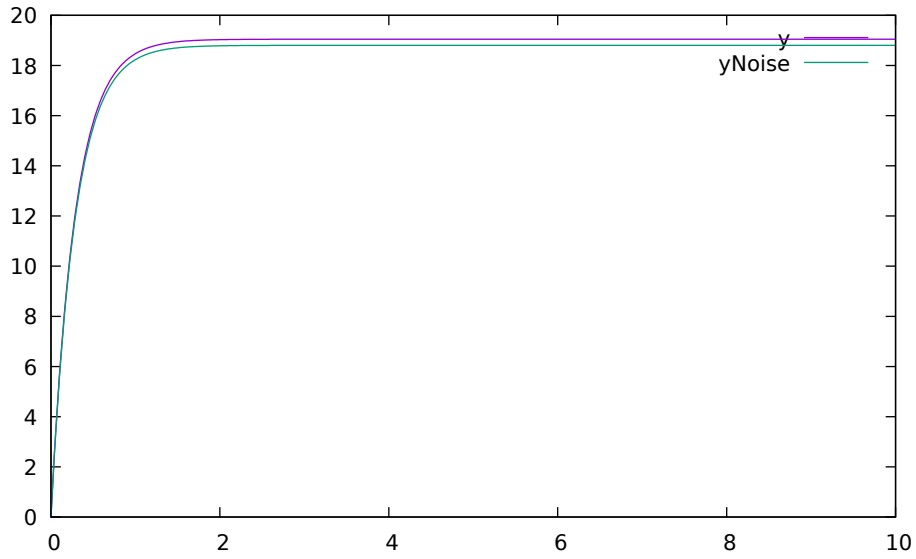


Figure 10.9: Closed loop control example.

10.3.2 Mathematical Modeling with Characteristic Equations

In most systems the relation between the inputs and outputs can be described by a linear differential equation. Tearing apart the solution of the differential equation into homogenous and particular parts is an important technique taught to the students in engineering courses, also illustrated in Figure 10.10.

$$\frac{\partial^n y}{\partial t^n} + a_1 \frac{\partial^{n-1} y}{\partial t^{n-1}} + \ldots + a_n y = b_0 \frac{\partial^m u}{\partial t^m} + \ldots + b_{m-1} \frac{\partial u}{\partial t} + b_m u$$

Now let us examine a second order system:

$$\ddot{y} + a_1 \dot{y} + a_2 y = 1$$

```
model NegRoots
  Real y;
  Real der_y;
  parameter Real a1 = 3;
  parameter Real a2 = 2;
equation
  der_y = der(y);
  der(der_y) + a1*der_y + a2*y = 1;
end NegRoots;
```

Choosing different values for a_1 and a_2 leads to different behavior as shown in Figure 10.11 and Figure 10.12.

In the first example the values of a_1 and a_2 are chosen in such way that the characteristic equation has negative real roots and thereby a stable output response, see Figure 10.11.

```
>>> simulate(NegRoots, stopTime=10)
record SimulationResult
    resultFile = "<<DOCHOME>>/NegRoots_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 10.0, numberOfIntervals = 500,
 tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'NegRoots', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
```

(次のページに続く)

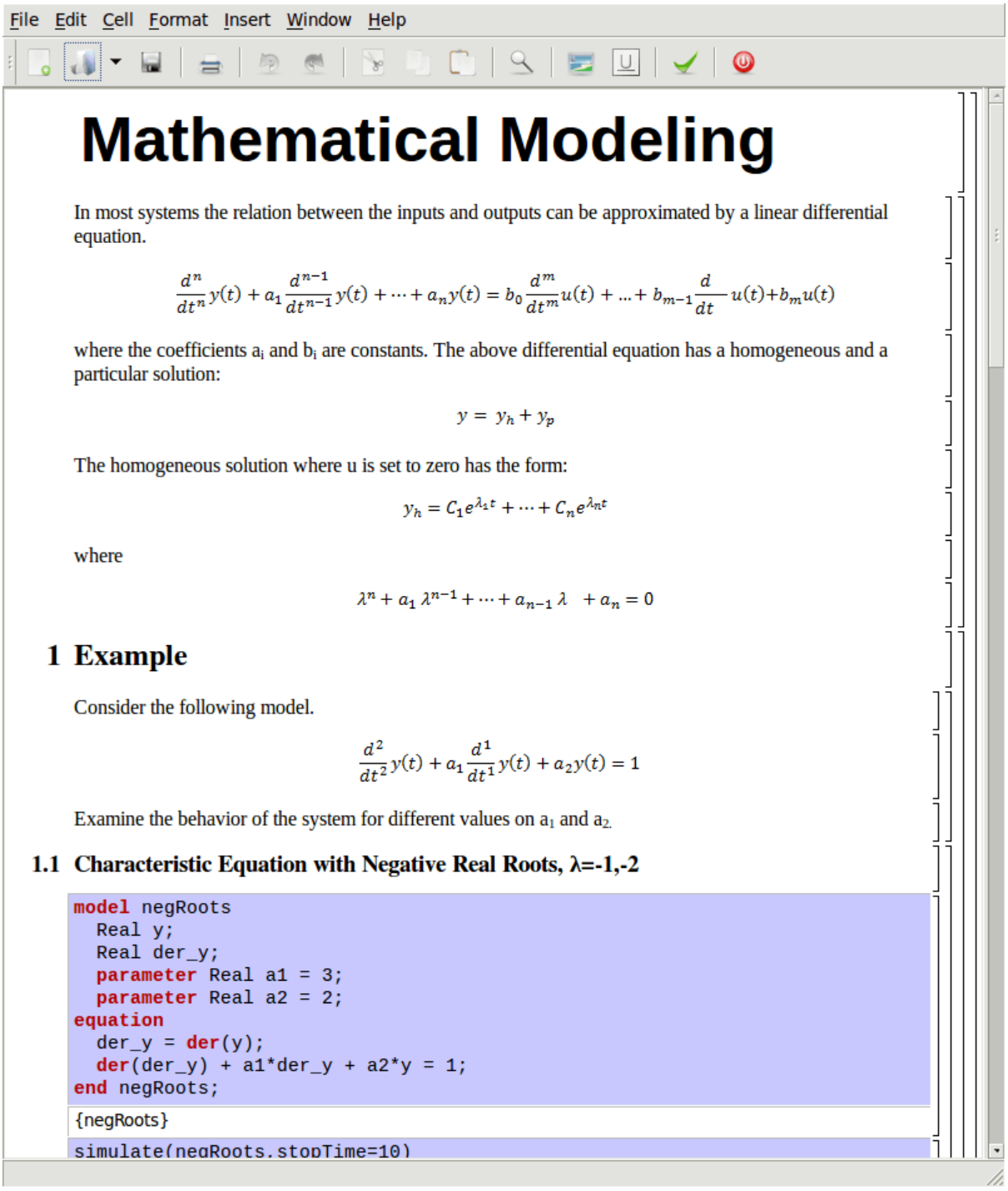


Figure 10.10: Mathematical modeling with characteristic equation.

(前のページからの続き)

```
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.5254291,
    timeBackend = 0.0028538,
    timeSimCode = 0.0008244000000000001,
    timeTtemplates = 0.0108581,
    timeCompile = 2.5259,
    timeSimulation = 0.1130366,
    timeTotal = 3.1807381
end SimulationResult;
```

警告:

Warning: The initial conditions are not fully specified. For more information set -d=initialization. In OMEdit Tools->Options->Simulation->OMCFlags, in OMNotebook call setCommandLineOptions("-d=initialization").

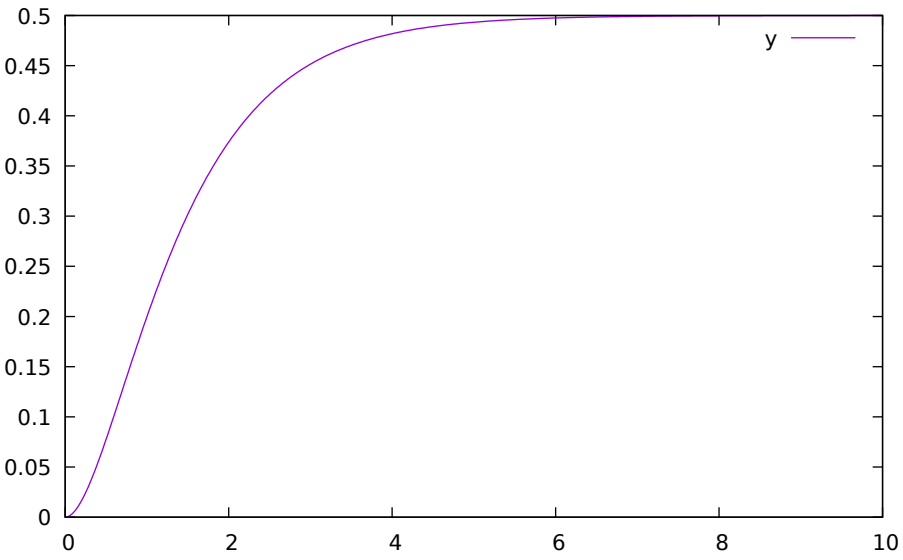


Figure 10.11: Characteristic equation with real negative roots.

The importance of the sign of the roots in the characteristic equation is illustrated in Figure 10.11 and Figure 10.12, e.g., a stable system with negative real roots and an unstable system with positive imaginary roots resulting in oscillations.

```
model ImgPosRoots
  Real y;
  Real der_y;
  parameter Real a1 = -2;
  parameter Real a2 = 10;
equation
  der_y = der(y);
  der(der_y) + a1*der_y + a2*y = 1;
end ImgPosRoots;
```

```
>>> simulate(ImgPosRoots, stopTime=10)
record SimulationResult
    resultFile = "<<DOCHOME>>/ImgPosRoots_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 10.0, numberOfIntervals = 500,
 tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'ImgPosRoots', options = '',
 outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.5066785,
    timeBackend = 0.0044695,
    timeSimCode = 0.0009482000000000001,
    timeTemplates = 0.0190591,
    timeCompile = 2.49831,
    timeSimulation = 0.1089323,
    timeTotal = 3.1401374
end SimulationResult;
```

警告:

Warning: The initial conditions are not fully specified. For more information set -d=initialization. In OMEdit Tools->Options->Simulation->OMCFlags, in OMNotebook call setCommandLineOptions("-d=initialization").

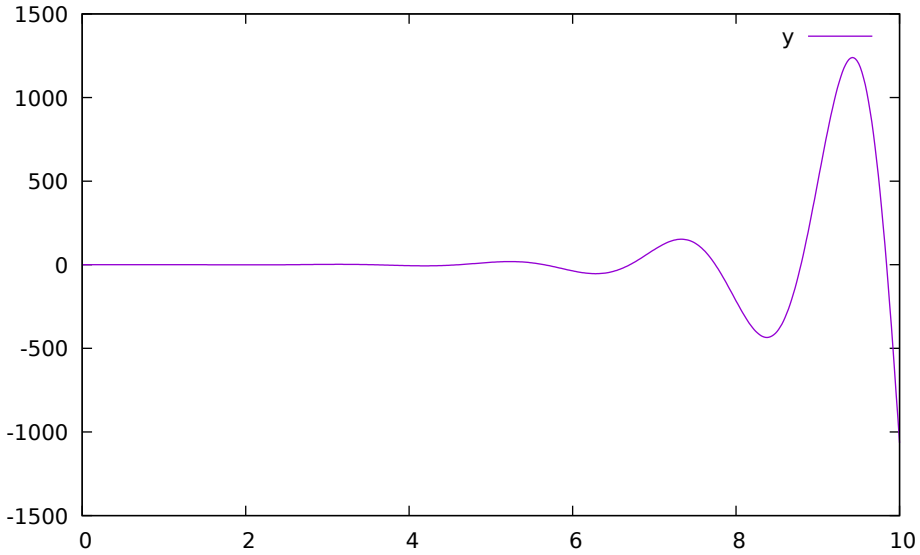


Figure 10.12: Characteristic equation with imaginary roots with positive real part.

The theory and application of Kalman filters is also explained in the interactive course material.

In reality noise is present in almost every physical system under study and therefore the concept of noise is also introduced in the course material, which is purely Modelica based.

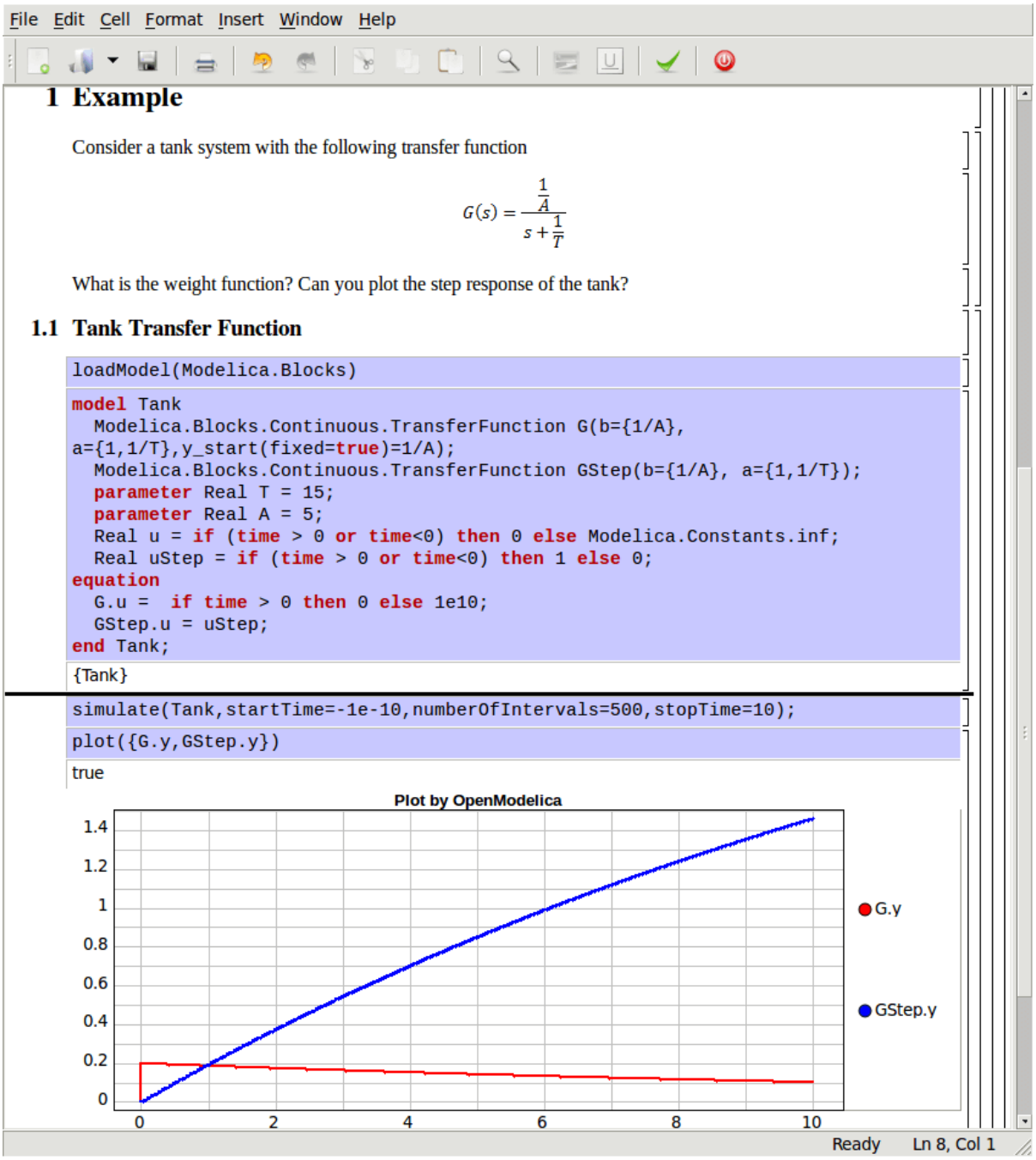


Figure 10.13: Step and pulse (weight function) response.

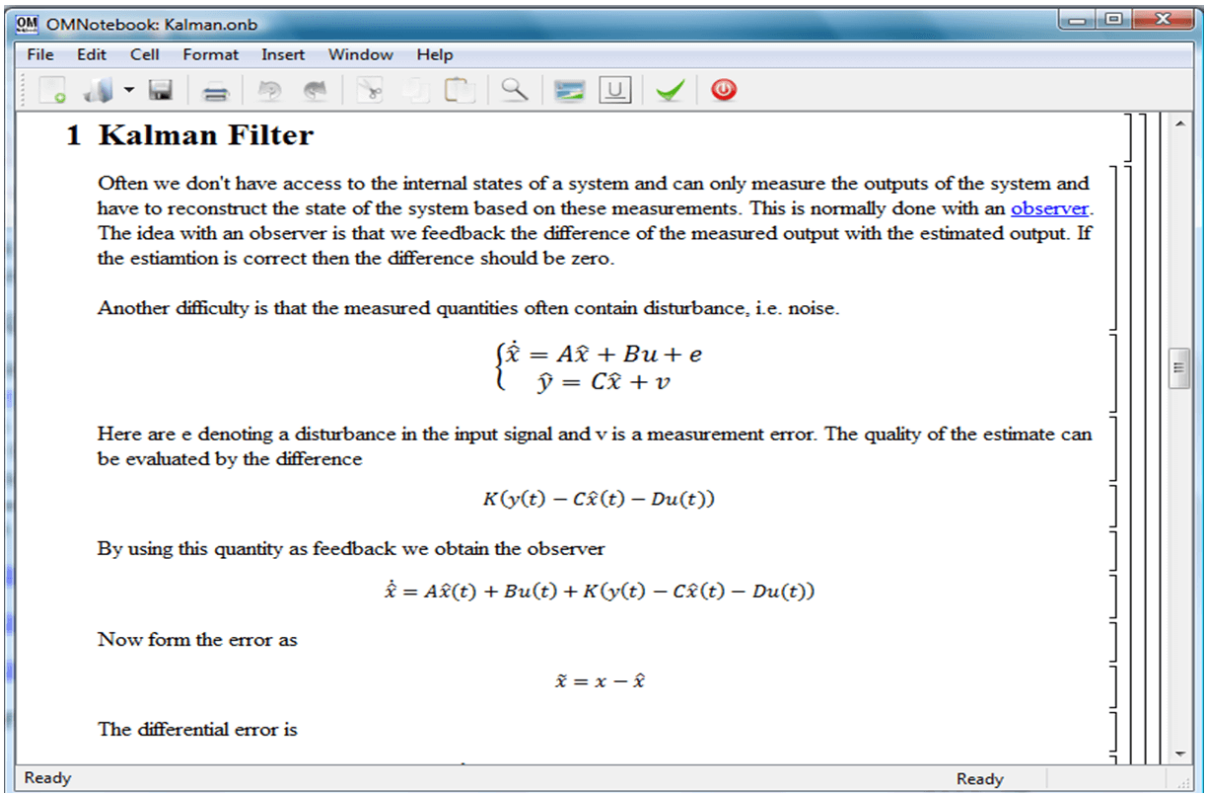


Figure 10.14: Theory background about Kalman filter.

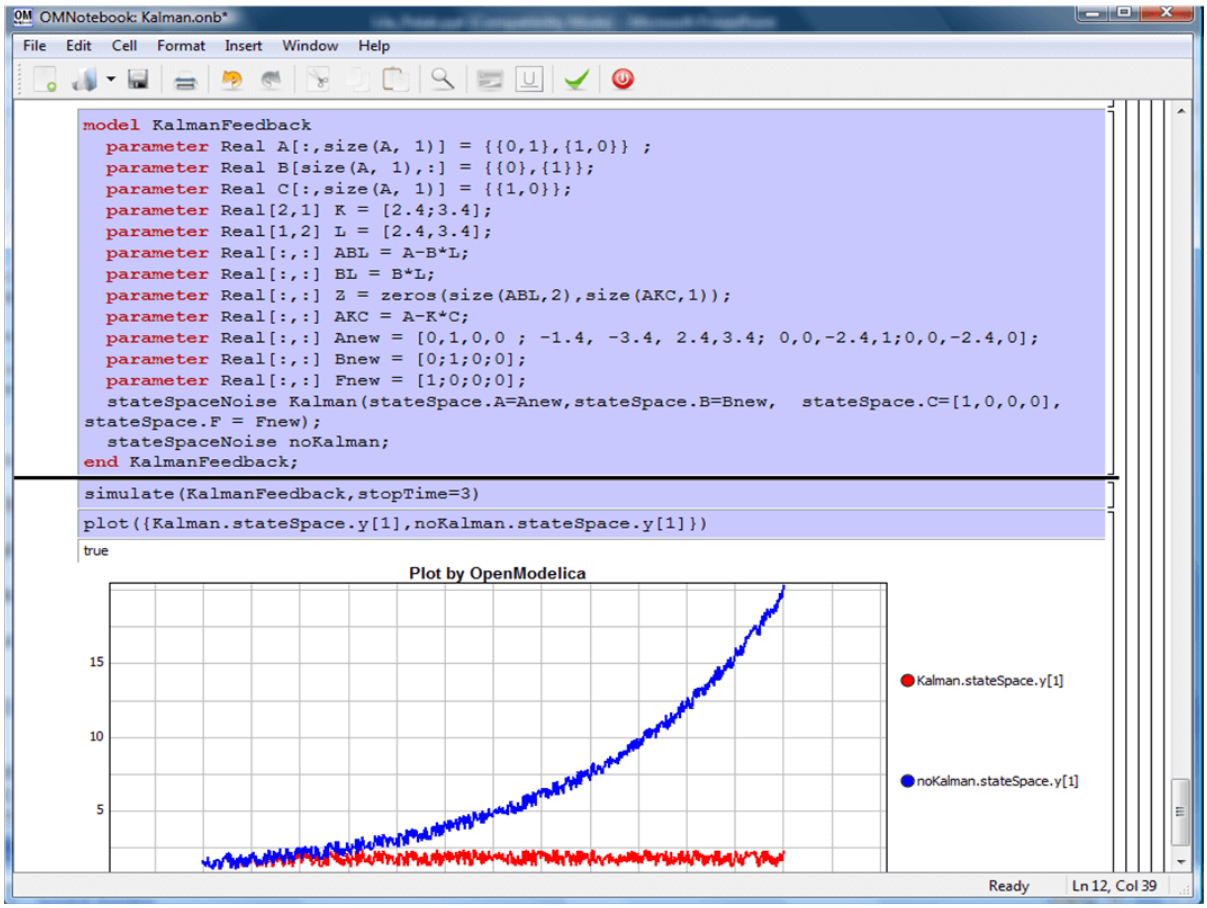


Figure 10.15: Comparison of a noisy system with feedback link in DrControl.

10.4 OpenModelica Notebook Commands

OMNotebook currently supports the commands and concepts that are described in this section.

10.4.1 Cells

Everything inside an OMNotebook document is made out of cells. A cell basically contains a chunk of data. That data can be text, images, or other cells. OMNotebook has four types of cells: headercell, textcell, inputcell, and groupcell. Cells are ordered in a tree structure, where one cell can be a parent to one or more additional cells. A tree view is available close to the right border in the notebook window to display the relation between the cells.

- ***Textcell* – This cell type is used to display ordinary text and** images. Each textcell has a style that specifies how text is displayed. The cell ´ s style can be changed in the menu Format->Styles, example of different styles are: Text, Title, and Subtitle. The Textcell type also has support for following links to other notebook documents.

- ***Inputcell* – This cell type has support for syntax highlighting and** evaluation. It is intended to be used for writing program code, e.g. Modelica code. Evaluation is done by pressing the key combination Shift+Return or Shift+Enter. All the text in the cell is sent to OMC (OpenModelica Compiler/interpreter), where the text is evaluated and the result is displayed below the inputcell. By double-clicking on the cell marker in the tree view, the inputcell can be collapsed causing the result to be hidden.

- ***Latexcell* – This cell type has support for evaluation of latex scripts.** It is intended to be mainly used for writing mathematical equations and formulas for advanced documentation in OMNotebook. Each Latexcell supports a maximum of one page document output.To evaluate this cell, latex must be installed in your system.The users can copy and paste the latex scripts and start the evaluation.Evaluation is done by pressing the key combination Shift+Return or Shift+Enter or the green color eval button present in the toolbar. The script in the cell is sent to latex compiler, where it is evaluated and the output is displayed hiding the latex source. By double-clicking on the cell marker in the tree view,the latex source is displayed for further modification.

- ***Groupcell* – This cell type is used to group together other cell. A** groupcell can be opened or closed. When a groupcell is opened all the cells inside the groupcell are visible, but when the groupcell is closed only the first cell inside the groupcell is visible. The state of the groupcell is changed by the user double-clicking on the cell marker in the tree view. When the groupcell is closed the marker is changed and the marker has an arrow at the bottom.

10.4.2 Cursors

An OMNotebook document contains cells which in turn contain text. Thus, two kinds of cursors are needed for positioning, text cursor and cell cursor:

- ***Textcursor* – A cursor between characters in a cell, appearing as a** small vertical line. Position the cursor by clicking on the text or using the arrow buttons.

- *Cellcursor* – **This cursor shows which cell currently has the input** focus. It consists of two parts. The main cellcursor is basically just a thin black horizontal line below the cell with input focus. The cellcursor is positioned by clicking on a cell, clicking between cells, or using the menu item Cell->Next Cell or Cell->Previous Cell. The cursor can also be moved with the key combination Ctrl+Up or Ctrl+Down. The dynamic cellcursor is a short blinking horizontal line. To make this visible, you must click once more on the main cellcursor (the long horizontal line). NOTE: In order to paste cells at the cellcursor, the *dynamic cellcursor must be made active* by clicking on the main cellcursor (the horizontal line).

10.4.3 Selection of Text or Cells

To perform operations on text or cells we often need to select a range of characters or cells.

- *Select characters* – **There are several ways of selecting characters,** e.g. double-clicking on a word, clicking and dragging the mouse, or click followed by a shift-click at an adjacent positioin selects the text between the previous click and the position of the most recent shift-click.
- *Select cells* – **Cells can be selected by clicking on them. Holding** down Ctrl and clicking on the cell markers in the tree view allows several cells to be selected, one at a time. Several cells can be selected at once in the tree view by holding down the Shift key. Holding down Shift selects all cells between last selected cell and the cell clicked on. This only works if both cells belong to the same groupcell.

10.4.4 File Menu

The following file related operations are available in the file menu:

- *Create a new notebook* – **A new notebook can be created using the** menu File->New or the key combination Ctrl+N. A new document window will then open, with a new document inside.
- *Open a notebook* – **To open a notebook use File->Open in the menu or** the key combination Ctrl+O. Only files of the type .onb or .nb can be opened. If a file does not follow the OMNotebook format or the FullForm Mathematica Notebook format, a message box is displayed telling the user what is wrong. Mathematica Notebooks must be converted to fullform before they can be opened in OMNotebook.
- *Save a notebook* – **To save a notebook use the menu item File->Save** or File->Save As. If the notebook has not been saved before the save as dialog is shown and a filename can be selected. OMNotebook can only save in xml format and the saved file is not compatible with Mathematica. Key combination for save is Ctrl+S and for save as Ctrl+Shift+S. The saved file by default obtains the file extension .onb.
- *Print* – **Printing a document to a printer is done by pressing the** key combination Ctrl+P or using the menu item File->Print. A normal print dialog is displayed where the usually properties can be changed.
- *Import old document* – **Old documents, saved with the old version of** OMNotebook where a different file format was used, can be opened using the menu item File->Import->Old OMNotebook file. Old documents have the extension .xml.

- *Export text* – **The text inside a document can be exported to a text** document. The text is exported to this document without almost any structure saved. The only structure that is saved is the cell structure. Each paragraph in the text document will contain text from one cell. To use the export function, use menu item File->Export->Pure Text.
- *Close a notebook window* – **A notebook window can be closed using the** menu item File->Close or the key combination Ctrl+F4. Any unsaved changes in the document are lost when the notebook window is closed.
- *Quitting OMNotebook* – **To quit OMNotebook, use menu item File->Quit** or the key combination Crtl+Q. This closes all notebook windows; users will have the option of closing OMC also. OMC will not automatically shutdown because other programs may still use it. Evaluating the command quit() has the same result as exiting OMNotebook.

10.4.5 Edit Menu

- *Editing cell text* – **Cells have a set of of basic editing functions.** The key combination for these are: Undo (Ctrl+Z), Redo (Ctrl+Y), Cut (Ctrl+X), Copy (Ctrl+C) and Paste (Ctrl+V). These functions can also be accessed from the edit menu; Undo (Edit->Undo), Redo (Edit->Redo), Cut (Edit->Cut), Copy (Edit->Copy) and Paste (Edit->Paste). Selection of text is done in the usual way by double-clicking, triple-clicking (select a paragraph), dragging the mouse, or using (Ctrl+A) to select all text within the cell.
- *Cut cell* – **Cells can be cut from a document with the menu item** Edit->Cut or the key combination Ctrl+X. The cut function will always cut cells if cells have been selected in the tree view, otherwise the cut function cuts text.
- *Copy cell* – **Cells can be copied from a document with the menu item** Edit->Copy or the key combination Ctrl+C. The copy function will always copy cells if cells have been selected in the tree view, otherwise the copy function copy text.
- *Paste cell* – **To paste copied or cut cells the cell cursor must be** selected in the location where the cells should be pasted. This is done by clicking on the cell cursor. Pasteing cells is done from the menu Edit->Paste or the key combination Ctrl+V. If the cell cursor is selected the paste function will always paste cells. OMNotebook share the same application-wide clipboard. Therefore cells that have been copied from one document can be pasted into another document. Only pointers to the copied or cut cells are added to the clipboard, thus the cell that should be pasted must still exist. Consequently a cell can not be pasted from a document that has been closed.
- *Find* – **Find text string in the current notebook, with the options** match full word, match cell, search within closed cells. Short command Ctrl+F.
- *Replace* – **Find and replace text string in the current notebook,** with the options match full word, match cell, search+replace within closed cells. Short command Ctrl+H.
- *View expression* – **Text in a cell is stored internally as a subset** of HTML code and the menu item Edit->View Expression let the user switch between viewing the text or the internal HTML representation. Changes made to the HTML code will affect how the text is displayed.

10.4.6 Cell Menu

- *Add textcell* – **A new textcell is added with the menu item Cell->Add** Cell (previous cell style) or the key combination Alt+Enter. The new textcell gets the same style as the previous selected cell had.
- *Add inputcell* – **A new inputcell is added with the menu item** Cell->Add Inputcell or the key combination Ctrl+Shift+I.
- *Add latexcell* – **A new latexcell is added with the menu item** Cell->Add Latexcell or the key combination Ctrl+Shift+E.
- *Add groupcell* – **A new groupcell is inserted with the menu item** Cell->Groupcell or the key combination Ctrl+Shift+G. The selected cell will then become the first cell inside the groupcell.
- *Ungroup groupcell* – **A groupcell can be ungrouped by selecting it in** the tree view and using the menu item Cell->Ungroup Groupcell or by using the key combination Ctrl+Shift+U. Only one groupcell at a time can be ungrouped.
- *Split cell* – **Spliting a cell is done with the menu item Cell->Split** cell or the key combination Ctrl+Shift+P. The cell is splited at the position of the text cursor.
- *Delete cell* – **The menu item Cell->Delete Cell will delete all cells** that have been selected in the tree view. If no cell is selected this action will delete the cell that have been selected by the cellcursor. This action can also be called with the key combination Ctrl+Shift+D or the key Del (only works when cells have been selected in the tree view).
- *Cellcursor* – **This cell type is a special type that shows which cell** that currently has the focus. The cell is basically just a thin black line. The cellcursor is moved by clicking on a cell or using the menu item Cell->Next Cell or Cell->Previous Cell. The cursor can also be moved with the key combination Ctrl+Up or Ctrl+Down.

10.4.7 Format Menu

- *Textcell* – **This cell type is used to display ordinary text and** images. Each textcell has a style that specifies how text is displayed. The cells style can be changed in the menu Format->Styles, examples of different styles are: Text, Title, and Subtitle. The Textcell type also have support for following links to other notebook documents.
- *Text manipulation* – **There are a number of different text** manipulations that can be done to change the appearance of the text. These manipulations include operations like: changing font, changing color and make text bold, but also operations like: changing the alignment of the text and the margin inside the cell. All text manipulations inside a cell can be done on single letters, words or the entire text. Text settings are found in the Format menu. The following text manipulations are available in OMNotebook:

> Font family

> Font face (Plain, Bold, Italic, Underline)

> Font size

> Font stretch

> Font color

> Text horizontal alignment

> Text vertical alignment

> Border thickness

> Margin (outside the border)

> Padding (inside the border)

10.4.8 Insert Menu

- ***Insert image* – Images are added to a document with the menu item** Insert->Image or the key combination Ctrl+Shift+M. After an image has been selected a dialog appears, where the size of the image can be chosen. The images actual size is the default value of the image. OMNotebook stretches the image accordantly to the selected size. All images are saved in the same file as the rest of the document.
- ***Insert link* – A document can contain links to other OMNotebook file** or Mathematica notebook and to add a new link a piece of text must first be selected. The selected text make up the part of the link that the user can click on. Inserting a link is done from the menu Insert->Link or with the key combination Ctrl+Shift+L. A dialog window, much like the one used to open documents, allows the user to choose the file that the link refers to. All links are saved in the document with a relative file path so documents that belong together easily can be moved from one place to another without the links failing.

10.4.9 Window Menu

- ***Change window* – Each opened document has its own document window.** To switch between those use the Window menu. The window menu lists all titles of the open documents, in the same order as they were opened. To switch to another document, simple click on the title of that document.

10.4.10 Help Menu

- ***About OMNotebook* – Accessing the about message box for OMNotebook** is done from the menu Help->About OMNotebook.
- ***About Qt* – To access the message box for Qt, use the menu** Help->About Qt.
- ***Help Text* – Opening the help text (document OMNotebookHelp.onb) for** OMNotebook can be done in the same way as any OMNotebook document is opened or with the menu Help->Help Text. The menu item can also be triggered with the key F1.

10.4.11 Additional Features

- *Links* – **By clicking on a link, OMNotebook will open the document** that is referred to in the link.
- *Update link* – **All links are stored with relative file path.** Therefore OMNotebook has functions that automatically updating links if a document is resaved in another folder. Every time a document is saved, OMNotebook checks if the document is saved in the same folder as last time. If the folder has changed, the links are updated.
- *Evaluate whole Notebook* – **All the cells present in the Notebook can** be evaluated in one step by pressing the red color evalall button in the toolbar. The cells are evaluated in the same order as they are in the Notebook.However the latexcells cannot be evaluated by this feature.
- *Evaluate several cells* – **Several inputcells can be evaluated at** the same time by selecting them in the treeview and then pressing the key combination Shift+Enter or Shift+Return. The cells are evaluated in the same order as they have been selected. If a groupcell is selected all inputcells in that groupcell are evaluated, in the order they are located in the groupcell.
- *Moving and Reordering cells in a Notebook* – **It is possible to shift cells** to a new position and change the hierarchical order of the document.This can be done by clicking the cell and press the Up and Down arrow button in the tool bar to move either Up or Down. The cells are moved one cell above or below.It is also possible to move a cell directly to a new position with one single click by pressing the red color bidirectional UpDown arrow button in the toolbar. To do this the user has to place the cell cursor to a position where the selected cells must be moved. After selecting the cell cursor position, select the cells you want to shift and press the bidirectional UpDown arrow button. The cells are shifted in the same order as they are selected.This is especially very useful when shifting a group cell.
- *Command completion* – **Inputcells have command completion support,** which checks if the user is typing a command (or any keyword defined in the file commands.xml) and finish the command. If the user types the first two or three letters in a command, the command completion function fills in the rest. To use command completion, press the key combination Ctrl+Space or Shift+Tab. The first command that matches the letters written will then appear. Holding down Shift and pressing Tab (alternative holding down Ctrl and pressing Space) again will display the second command that matches. Repeated request to use command completion will loop through all commands that match the letters written. When a command is displayed by the command completion functionality any field inside the command that should be edited by the user is automatically selected. Some commands can have several of these fields and by pressing the key combination Ctrl+Tab, the next field will be selected inside the command. > Active Command completion: Ctrl+Space / Shift+Tab > Next command: Ctrl+Space / Shift+Tab > Next field in command: Ctrl+Tab'
- *Generated plot* – **When plotting a simulation result, OMC uses the** program Ptplot to create a plot. From Ptplot OMNotebook gets an image of the plot and automatically adds that image to the output part of an inputcell. Like all other images in a document, the plot is saved in the document file when the document is saved.
- *Stylesheet* –**OMNotebook follows the style settings defined in** stylesheet.xml and the correct style is applied to a cell when the cell is created.
- *Automatic Chapter Numbering* – **OMNotebook automatically numbers** different chapter, subchapter, section and other styles. The user can specify which styles should have chapter numbers and which

level the style should have. This is done in the stylesheet.xml file. Every style can have a <chapterLevel> tag that specifies the chapter level. Level 0 or no tag at all, means that the style should not have any chapter numbering.

- ***Scrollarea* – Scrolling through a document can be done by using the** mouse wheel. A document can also be scrolled by moving the cell cursor up or down.
- ***Syntax highlighter* – The syntax highlighter runs in a separated** thread which speeds up the loading of large document that contains many Modelica code cells. The syntax highlighter only highlights when letters are added, not when they are removed. The color settings for the different types of keywords are stored in the file modelicacolors.xml. Besides defining the text color and background color of keywords, whether or not the keywords should be bold or/and italic can be defined.
- ***Change indicator* – A star (*) will appear behind the filename in** the title of notebook window if the document has been changed and needs saving. When the user closes a document that has some unsaved change, OMNotebook asks the user if he/she wants to save the document before closing. If the document never has been saved before, the save-as dialog appears so that a filename can be choosen for the new document.
- ***Update menus* – All menus are constantly updated so that only menu** items that are linked to actions that can be performed on the currently selected cell is enabled. All other menu items will be disabled. When a textcell is selected the Format menu is updated so that it indicates the text settings for the text, in the current cursor position.

10.5 References

課題: Add these into extrarefs.bib and cite them somewhere

Eric Allen, Robert Cartwright, Brian Stoler. DrJava: A lightweight pedagogic environment for Java. In Proceedings of the 33rd ACM Technical Symposium on Computer Science Education (SIGCSE 2002) (Northern Kentucky – The Southern Side of Cincinnati, USA, February 27 – March 3, 2002).

Anders Fernström, Ingemar Axelsson, Peter Fritzson, Anders Sandholm, Adrian Pop. OMNotebook – Interactive WYSIWYG Book Software for Teaching Programming. In Proc. of the Workshop on Developing Computer Science Education – How Can It Be Done?. Linköping University, Dept. Computer & Inf. Science, Linköping, Sweden, March 10, 2006.

Eva-Lena Lengquist-Sandelin, Susanna Monemar, Peter Fritzson, and Peter Bunus. DrModelica – A Web-Based Teaching Environment for Modelica. In Proceedings of the 44th Scandinavian Conference on Simulation and Modeling (SIMS’2003), available at www.scan-sims.org. Västerås, Sweden. September 18-19, 2003.

第11章　Optimization with OpenModelica

The following facilities for model-based optimization are provided with OpenModelica:

- ***Builtin Dynamic Optimization with OpenModelica and IpOpt*** **using** dynamic optimization is the recommended way of performing dynamic optimization with OpenModelica.
- ***Dynamic Optimization with OpenModelica and CasADi*****. Use this if you want to employ** the CasADi tool for dynamic optimization.
- **Classical** ***Parameter Sweep Optimization using OMOptim*****. Use** this if you have a static optimization problem.

11.1 Builtin Dynamic Optimization with OpenModelica and IpOpt

Note: this is a very short preliminary decription which soon will be considerably improved.

OpenModelica provides builtin dynamic optimization of models by using the powerful symbolic machinery of the OpenModelica compiler for more efficient and automatic solution of dynamic optimization problems.

The builtin dynamic optimization allows users to define optimal control problems (OCP) using the Modelica language for the model and the optimization language extension called Optimica (currently partially supported) for the optimization part of the problem. This is used to solve the underlying dynamic optimization model formulation using collocation methods, using a single execution instead of multiple simulations as in the parameter-sweep optimization described in section *Parameter Sweep Optimization using OMOptim*.

For more detailed information regarding background and methods, see [BOR+12][RBB+14]

11.2 Compiling the Modelica code

Before starting the optimization the model should be symbolically instantiated by the compiler in order to get a single flat system of equations. The model variables should also be scalarized. The compiler frontend performs this, including syntax checking, semantics and type checking, simplification and constant evaluation etc. are applied. Then the complete flattened model can be used for initialization, simulation and last but not least for model-based dynamic optimization.

The OpenModelica command optimize(ModelName) from OMShell, OMNotebook or MDT runs immediately the optimization. The generated result file can be read in and visualized with OMEdit or within OMNotebook.

11.3 An Example

In this section, a simple optimal control problem will be solved. When formulating the optimization problems, models are expressed in the Modelica language and optimization specifications. The optimization language specification allows users to formulate dynamic optimization problems to be solved by a numerical algorithm. It includes several constructs including a new specialized class optimization, a constraint section, startTime, finalTime etc. See the optimal control problem for batch reactor model below.

Create a new file named *BatchReactor.mo* and save it in you working directory. Notice that this model contains both the dynamic system to be optimized and the optimization specification.

Once we have formulated the undelying optimal control problems, we can run the optimization by using OMShell, OMNotebook, MDT, OMEdit using command line terminals similar to the options described below:

```
>>> setCommandLineOptions("-g=Optimica");
```

Listing 11.1: BatchReactor.mo

```
model BatchReactor
  Real x1(start =1, fixed=true, min=0, max=1);
  Real x2(start =0, fixed=true, min=0, max=1);
  input Real u(min=0, max=5);
equation
  der(x1) = -(u+u^2/2)*x1;
  der(x2) = u*x1;
end BatchReactor;
```

```
optimization nmpcBatchReactor(objective=-x2)
  extends BatchReactor;
end nmpcBatchReactor;
```

```
>>> optimize(nmpcBatchReactor, numberOfIntervals=16, stopTime=1, tolerance=1e-8)
record SimulationResult
    resultFile = "<<DOCHOME>>/nmpcBatchReactor_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 16,␣
↪tolerance = 1e-08, method = 'optimization', fileNamePrefix = 'nmpcBatchReactor',␣
↪options = '', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags␣
↪= ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.

Optimizer Variables
========================================================
State[0]:x1(start = 1, nominal = 1, min = 0, max = 1, init = 1)
State[1]:x2(start = 0, nominal = 1, min = 0, max = 1, init = 0)
Input[2]:u(start = 0, nominal = 5, min = 0, max = 5)
--------------------------------------------------------
number of nonlinear constraints: 0
========================================================

******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************
```

(次のページに続く)

(前のページからの続き)

```
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.4291655,
    timeBackend = 0.0268885,
    timeSimCode = 0.0055183,
    timeTemplates = 0.0157422,
    timeCompile = 2.5454953,
    timeSimulation = 0.231716,
    timeTotal = 3.2565022
end SimulationResult;
```

The control and state trajectories of the optimization results:

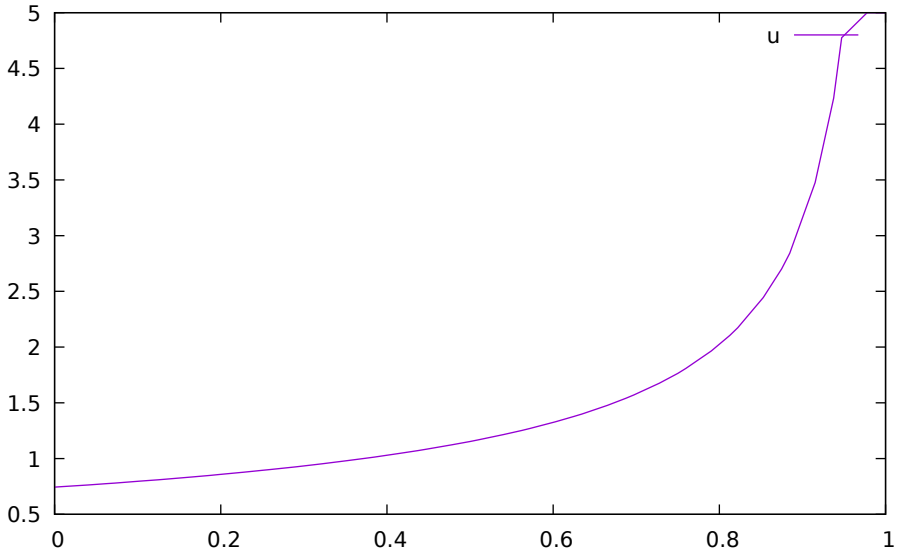


Figure 11.1: Optimization results for Batch Reactor model – input variables.

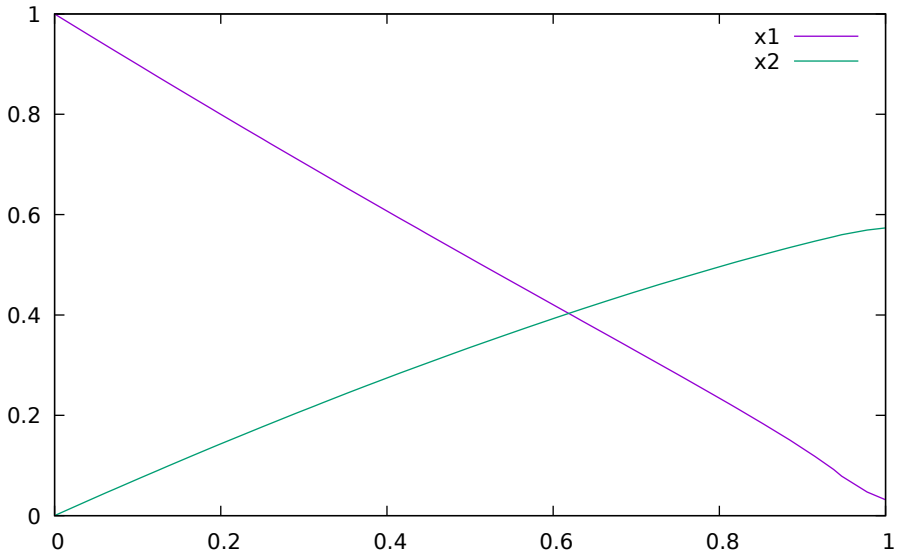


Figure 11.2: Optimization results for Batch Reactor model – state variables.

11.4 Different Options for the Optimizer IPOPT

Table 11.1: New meanings of the usual simualtion options for Ipopt.

numberOfIntervals		collocation intervals
startTime, stopTime		time horizon
tolerance = 1e-8	e.g. 1e-8	solver tolerance
simflags	all run/debug options	

Table 11.2: New simulation options for Ipopt.

-lv	LOG_IPOPT	console output
-ipopt_hesse	CONST,BFGS,NUM	hessian approximation
-ipopt_max_iter	number e.g. 10	maximal number of iteration for ipopt
externalInput.csv		input guess

11.5 Dynamic Optimization with OpenModelica and CasADi

OpenModelica coupling with CasADi supports dynamic optimization of models by OpenModelica exporting the optimization problem to CasADi which performs the optimization. In order to convey the dynamic system model information between Modelica and CasADi, we use an XML-based model exchange format for differential-algebraic equations (DAE). OpenModelica supports export of models written in Modelica and the Optimization language extension using this XML format, while CasADi supports import of models represented in this format. This allows users to define optimal control problems (OCP) using Modelica and Optimization language specifications, and solve the underlying model formulation using a range of optimization methods, including direct collocation and direct multiple shooting.

11.5.1 Compiling the Modelica code

Before exporting a model to XML, the model should be symbolically instantiated by the compiler in order to get a single flat system of equations. The model variables should also be scalarized. The compiler frontend performs this, including syntax checking, semantics and type checking, simplification and constant evaluation etc. are applied. Then the complete flattened model is exported to XML code. The exported XML document can then be imported to CasADi for model-based dynamic optimization.

The OpenModelica command translateModelXML(ModelName) from OMShell, OMNotebook or MDT exports the XML. The export XML command is also integrated with OMEdit. Select XML > Export XML the XML document is generated in the current directory of omc. You can use the cd() command to see the current location. After the command execution is complete you will see that a file ModelName.xml has been exported.

Assuming that the model is defined in the modelName.mo, the model can also be exported to an XML code using the following steps from the terminal window:

- Go to the path where your model file found
- Run command omc -g=Optimica --simCodeTarget=XML Model.mo

11.5.2 An example

In this section, a simple optimal control problem will be solved. When formulating the optimization problems, models are expressed in the Modelica language and optimization specifications. The optimization language specification allows users to formulate dynamic optimization problems to be solved by a numerical algorithm. It includes several constructs including a new specialized class optimization, a constraint section, startTime, finalTime etc. See the optimal control problem for batch reactor model below.

Create a new file named *BatchReactor.mo* and save it in you working directory. Notice that this model contains both the dynamic system to be optimized and the optimization specification.

```
>>> list(BatchReactor)
model BatchReactor
  Real x1(start = 1, fixed = true, min = 0, max = 1);
  Real x2(start = 0, fixed = true, min = 0, max = 1);
  input Real u(min = 0, max = 5);
equation
  der(x1) = -(u + u ^ 2 / 2) * x1;
  der(x2) = u * x1;
end BatchReactor;
```

One we have formulated the undelying optimal control problems, we can export the XML by using OMShell, OMNotebook, MDT, OMEdit or command line terminals which are described in Section *XML Import to CasADi via OpenModelica Python Script*.

To export XML, we set the simulation target to XML:

```
>>> translateModelXML(BatchReactor)
"<<DOCHOME>>/BatchReactor.xml"
```

This will generate an XML file named *BatchReactor.xml* (Listing 11.2) that contains a symbolic representation of the optimal control problem and can be inspected in a standard XML editor.

Listing 11.2: BatchReactor.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<OpenModelicaModelDescription
  xmlns:exp="https://svn.jmodelica.org/trunk/XML/daeExpressions.xsd"
  xmlns:equ="https://svn.jmodelica.org/trunk/XML/daeEquations.xsd"
  xmlns:fun="https://svn.jmodelica.org/trunk/XML/daeFunctions.xsd"
  xmlns:opt="https://svn.jmodelica.org/trunk/XML/daeOptimization.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  fmiVersion="1.0"
  modelName="BatchReactor"
  modelIdentifier="BatchReactor"
  guid="{7b5fc200-fbaf-4f79-991d-36774bc10c6d}"
  generationDateAndTime="2020-12-14T00:19:16"
```

(次のページに続く)

(前のページからの続き)

```
  variableNamingConvention="structured"
  numberOfContinuousStates="2"
  numberOfEventIndicators="0"
  >

  <VendorAnnotations>
    <Tool name="OpenModelica Compiler OMCompiler v1.14.1"> </Tool>
  </VendorAnnotations>


  <ModelVariables>
    <ScalarVariable name="x1"  valueReference="0" variability="continuous"␣
↪causality="internal" alias="noAlias">
      <Real start="1.0" fixed="true" min="0.0" max="1.0"  />
      <QualifiedName>
        <exp:QualifiedNamePart name="x1"/>
      </QualifiedName>
    <isLinearTimedVariables>
      <TimePoint index="0" isLinear="true"/>
    </isLinearTimedVariables>
      <VariableCategory>state</VariableCategory>
    </ScalarVariable>

    <ScalarVariable name="x2"  valueReference="1" variability="continuous"␣
↪causality="internal" alias="noAlias">
      <Real start="0.0" fixed="true" min="0.0" max="1.0"  />
      <QualifiedName>
        <exp:QualifiedNamePart name="x2"/>
      </QualifiedName>
    <isLinearTimedVariables>
      <TimePoint index="0" isLinear="true"/>
    </isLinearTimedVariables>
      <VariableCategory>state</VariableCategory>
    </ScalarVariable>
    <ScalarVariable name="der(x1)"  valueReference="2" variability="continuous"␣
↪causality="internal" alias="noAlias">
      <Real     />
      <QualifiedName>
        <exp:QualifiedNamePart name="x1"/>
      </QualifiedName>
    <isLinearTimedVariables>
      <TimePoint index="0" isLinear="true"/>
    </isLinearTimedVariables>
      <VariableCategory>derivative</VariableCategory>
    </ScalarVariable>

    <ScalarVariable name="der(x2)"  valueReference="3" variability="continuous"␣
↪causality="internal" alias="noAlias">
      <Real     />
      <QualifiedName>
        <exp:QualifiedNamePart name="x2"/>
      </QualifiedName>
    <isLinearTimedVariables>
      <TimePoint index="0" isLinear="true"/>
    </isLinearTimedVariables>
      <VariableCategory>derivative</VariableCategory>
    </ScalarVariable>
    <ScalarVariable name="u"  valueReference="4" variability="continuous"␣
↪causality="input" alias="noAlias">
      <Real  min="0.0" max="5.0"  />
      <QualifiedName>
```

(次のページに続く)

(前のページからの続き)

```
        <exp:QualifiedNamePart name="u"/>
      </QualifiedName>
    <isLinearTimedVariables>
      <TimePoint index="0" isLinear="true"/>
    </isLinearTimedVariables>
      <VariableCategory>algebraic</VariableCategory>
    </ScalarVariable>
  </ModelVariables>

  <equ:BindingEquations>
  </equ:BindingEquations>

  <equ:DynamicEquations>
    <equ:Equation>
      <exp:Sub>
        <exp:Der>
          <exp:Identifier>
            <exp:QualifiedNamePart name="x2"/>
          </exp:Identifier>
        </exp:Der>
        <exp:Mul>
          <exp:Identifier>
            <exp:QualifiedNamePart name="u"/>
          </exp:Identifier>
          <exp:Identifier>
            <exp:QualifiedNamePart name="x1"/>
          </exp:Identifier>
        </exp:Mul>
      </exp:Sub>
    </equ:Equation>
    <equ:Equation>
      <exp:Sub>
        <exp:Der>
          <exp:Identifier>
            <exp:QualifiedNamePart name="x1"/>
          </exp:Identifier>
        </exp:Der>
        <exp:Mul>
          <exp:Sub>
            <exp:Mul>
              <exp:RealLiteral>-0.5</exp:RealLiteral>
              <exp:Pow>
                <exp:Identifier>
                  <exp:QualifiedNamePart name="u"/>
                </exp:Identifier>
                <exp:RealLiteral>2.0</exp:RealLiteral>
              </exp:Pow>
            </exp:Mul>
            <exp:Identifier>
              <exp:QualifiedNamePart name="u"/>
            </exp:Identifier>
          </exp:Sub>
          <exp:Identifier>
            <exp:QualifiedNamePart name="x1"/>
          </exp:Identifier>
        </exp:Mul>
      </exp:Sub>
    </equ:Equation>
  </equ:DynamicEquations>

  <equ:InitialEquations>
```

(次のページに続く)

(前のページからの続き)

```
    <equ:Equation>
      <exp:Sub>
        <exp:Identifier>
          <exp:QualifiedNamePart name="x1"/>
        </exp:Identifier>
        <exp:RealLiteral>1.0</exp:RealLiteral>
      </exp:Sub>
    </equ:Equation>

    <equ:Equation>
      <exp:Sub>
        <exp:Identifier>
          <exp:QualifiedNamePart name="x2"/>
        </exp:Identifier>
        <exp:RealLiteral>0.0</exp:RealLiteral>
      </exp:Sub>
    </equ:Equation>
    <equ:Equation>
      <exp:Sub>
        <exp:Identifier>
          <exp:QualifiedNamePart name="x1"/>
        </exp:Identifier>
        <exp:Identifier>
          <exp:QualifiedNamePart name="$START"/>
          <exp:QualifiedNamePart name="x1"/>
        </exp:Identifier>
      </exp:Sub>
    </equ:Equation>
    <equ:Equation>
      <exp:Sub>

      </exp:Sub>
    </equ:Equation>
    <equ:Equation>
      <exp:Sub>

      </exp:Sub>
    </equ:Equation>
    <equ:Equation>
      <exp:Sub>
        <exp:Identifier>
          <exp:QualifiedNamePart name="x2"/>
        </exp:Identifier>
        <exp:Identifier>
          <exp:QualifiedNamePart name="$START"/>
          <exp:QualifiedNamePart name="x2"/>
        </exp:Identifier>
      </exp:Sub>
    </equ:Equation>
  </equ:InitialEquations>

  <fun:Algorithm>
  </fun:Algorithm>

  <fun:RecordsList>
  </fun:RecordsList>

  <fun:FunctionsList>
  </fun:FunctionsList>

  <opt:Optimization>
```

(次のページに続く)

(前のページからの続き)

```
    <opt:TimePoints>
      <opt:TimePoint  >
      </opt:TimePoint>
    </opt:TimePoints>
    <opt:PathConstraints>
    </opt:PathConstraints>
  </opt:Optimization>

</OpenModelicaModelDescription>
```

11.5.3 XML Import to CasADi via OpenModelica Python Script

The symbolic optimal control problem representation (or just model description) contained in BatchReactor.xml can be imported into CasADi in the form of the SymbolicOCP class via OpenModelica python script.

The SymbolicOCP class contains symbolic representation of the optimal control problem designed to be general and allow manipulation. For a more detailed description of this class and its functionalities, we refer to the API documentation of CasADi.

The following step compiles the model to an XML format, imports to CasADi and solves an optimization problem in windows PowerShell:

1. Create a new file named BatchReactor.mo and save it in you working directory.

   E.g. C:\OpenModelica1.9.2\share\casadi\testmodel

1. Perform compilation and generate the XML file

   a. Go to your working directory

   E.g. cd C:\OpenModelica1.9.2\share\casadi\testmodel

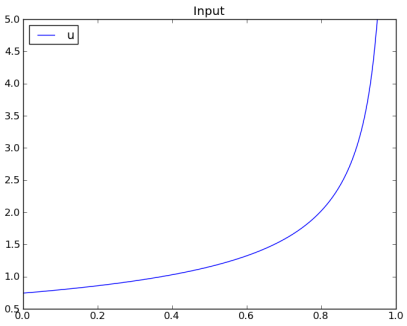
a. Go to omc path from working directory and run the following command

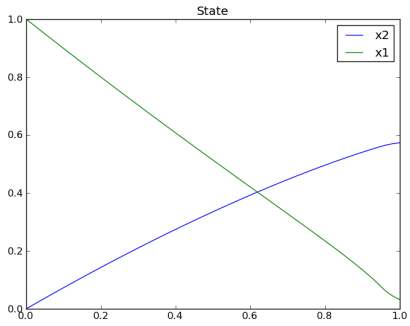
   E.g. ..\..\..\bin\omc +s -g=Optimica --simCodeTarget=XML BatchReactor.mo

3. Run defaultStart.py python script from OpenModelica optimization directory

E.g. Python.exe ..\share\casadi\scripts defaultStart.py BatchReactor.xml

The control and state trajectories of the optimization results are shown below:





11.6 Parameter Sweep Optimization using OMOptim

OMOptim is a tool for parameter sweep design optimization of Modelica models. By optimization, one should understand a procedure which minimizes/maximizes one or more objective functions by adjusting one or more parameters. This is done by the optimization algorithm performing a parameter swep, i.e., systematically adjusting values of selected parameters and running a number of simulations for different parameter combinations to find a parameter setting that gives an optimal value of the goal function.

OMOptim 0.9 contains meta-heuristic optimization algorithms which allow optimizing all sorts of models with following functionalities:

- One or several objectives optimized simultaneously
- One or several parameters (integer or real variables)

However, the user must be aware of the large number of simulations an optimization might require.

11.6.1 Preparing the Model

Before launching OMOptim, one must prepare the model in order to optimize it.

Parameters

An optimization parameter is picked up from all model variables. The choice of parameters can be done using the OMOptim interface.

For all intended parameters, please note that:

- **The corresponding variable is constant during all simulations.** The OMOptim optimization in version 0.9 only concerns static parameters' optimization *i.e.* values found for these parameters will be constant during all simulation time.
- **The corresponding variable should play an input role in the model** *i.e.* its modification influences model simulation results.

Constraints

If some constraints should be respected during optimization, they must be defined in the Modelica model itself.

For instance, if mechanical stress must be less than 5 $N.m^{-2}$, one should write in the model:

```
assert(mechanicalStress < 5, "Mechanical stress too high");
```

If during simulation, the variable *mechanicalStress* exceeds 5 $N.m^{-2}$, the simulation will stop and be considered as a failure.

Objectives

As parameters, objectives are picked up from model variables. Objectives' values are considered by the optimizer at the *final time*.

11.6.2 Set problem in OMOptim

Launch OMOptim

OMOptim can be launched using the executable placed in OpenModelicaInstallationDirectory/bin/ OMOptim/OMOptim.exe. Alternately, choose OpenModelica > OMOptim from the start menu.

Create a new project

To create a new project, click on menu File -> New project

Then set a name to the project and save it in a dedicated folder. The created file created has a .min extension. It will contain information regarding model, problems, and results loaded.

Load models

First, you need to load the model(s) you want to optimize. To do so, click on *Add .mo* button on main window or select menu *Model -> Load Mo file*⋯

When selecting a model, the file will be loaded in OpenModelica which runs in the background.

While OpenModelica is loading the model, you could have a frozen interface. This is due to multi-threading limitation but the delay should be short (few seconds).

You can load as many models as you want.

If an error occurs (indicated in log window), this might be because:

- Dependencies have not been loaded before (e.g. modelica library)
- Model use syntax incompatible with OpenModelica.

Dependencies

OMOptim should detect dependencies and load corresponding files. However, it some errors occur, please load by yourself dependencies. You can also load Modelica library using Model->Load Modelica library.

When the model correctly loaded, you should see a window similar to Figure 11.3.

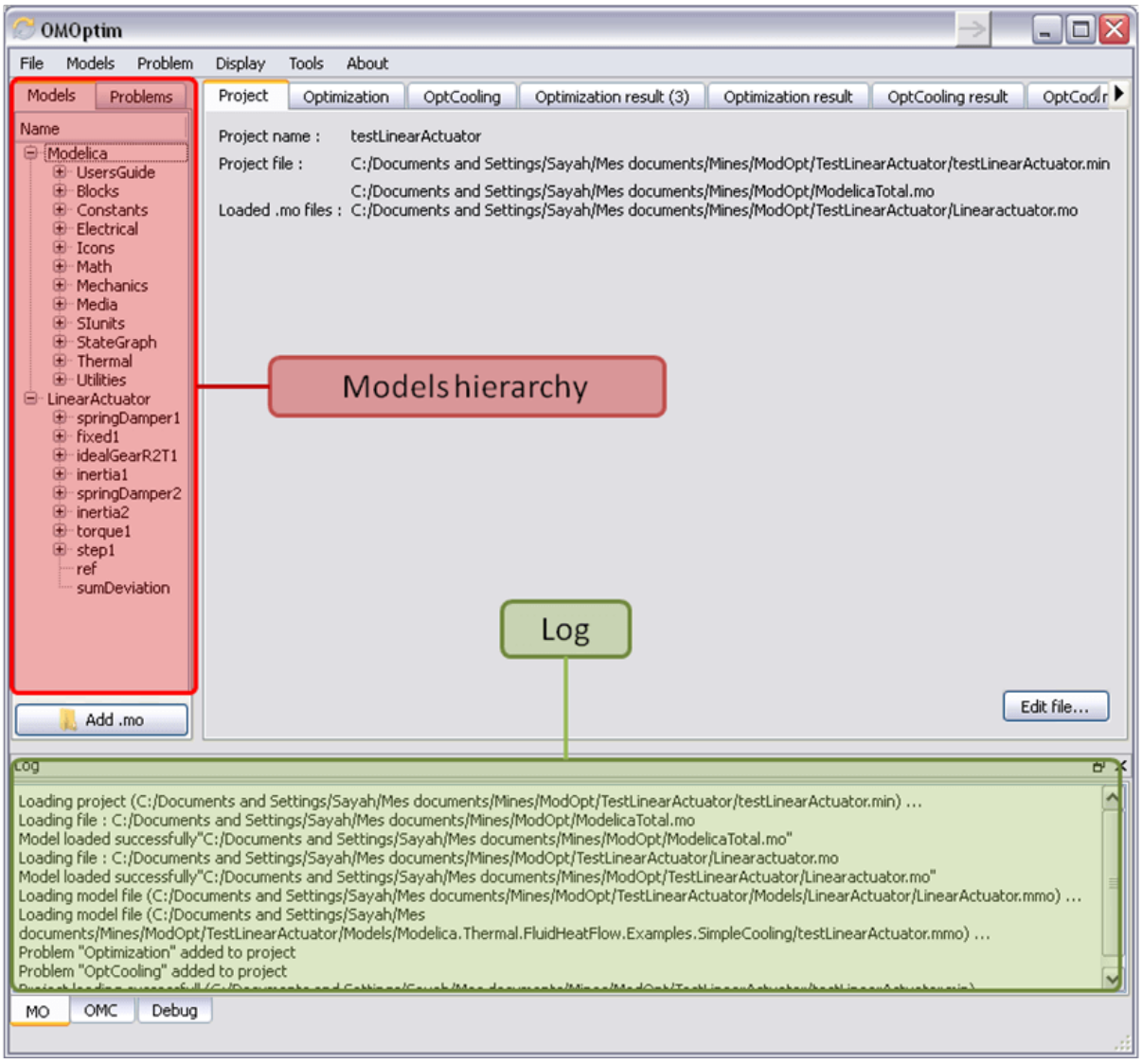


Figure 11.3: OMOptim window after having loaded model.

Create a new optimization problem

Problem->Add Problem->Optimization

A dialog should appear. Select the model you want to optimize. Only Model can be selected (no Package, Component, Block⋯).

A new form will be displayed. This form has two tabs. One is called Variables, the other is called Optimization.

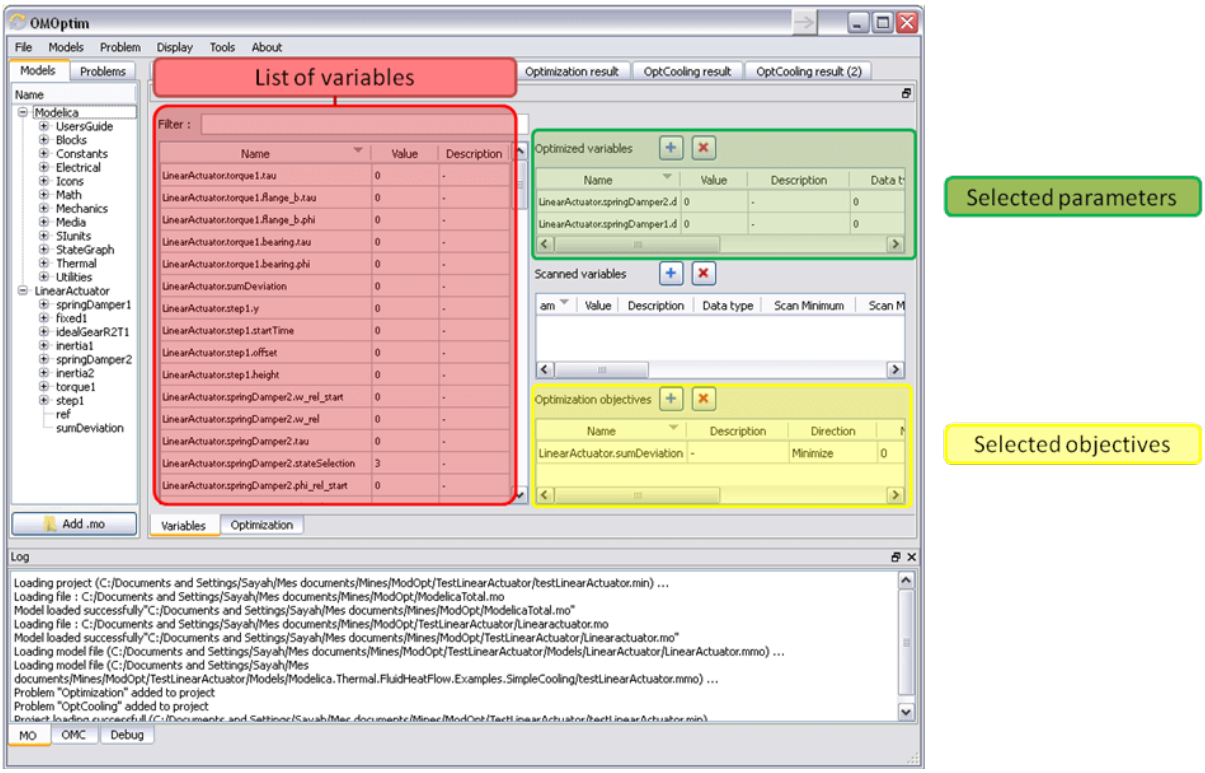


Figure 11.4: Forms for defining a new optimization problem.

List of Variables is Empty

If variables are not displayed, right click on model name in model hierarchy, and select *Read variables*.

Select Optimized Variables

To set optimization, we first have to define the variables the optimizer will consider as free *i.e.* those that it should find best values of. To do this, select in the left list, the variables concerned. Then, add them to *Optimized variables* by clicking on corresponding button (✚).

For each variable, you must set minimum and maximum values it can take. This can be done in the *Optimized variables* table.

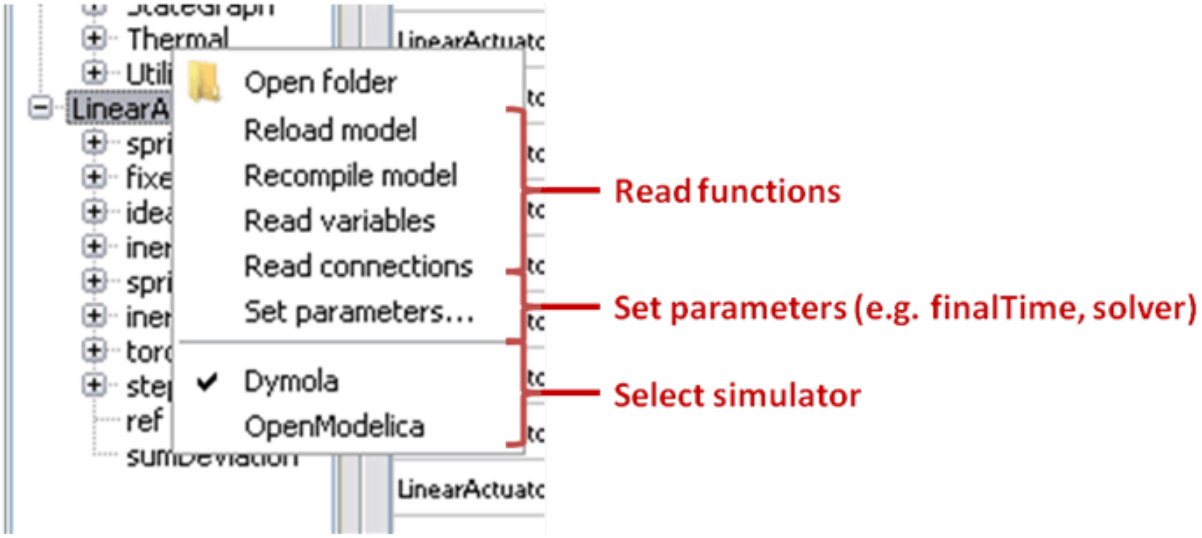


Figure 11.5: Selecting read variables, set parameters, and selecting simulator.

Select objectives

Objectives correspond to the final values of chosen variables. To select these last, select in left list variables concerned and click ✚ button of *Optimization objectives* table.

For each objective, you must:

- **Set minimum and maximum values it can take. If a configuration does** not respect these values, this configuration won't be considered. You also can set minimum and maximum equals to "- " : it will then
- Define whether objective should be minimized or maximized.

This can be done in the *Optimized variables* table.

Select and configure algorithm

After having selected variables and objectives, you should now select and configure optimization algorithm. To do this, click on *Optimization* tab.

Here, you can select optimization algorithm you want to use. In version 0.9, OMOptim offers three different genetic algorithms. Let's for example choose SPEA2Adapt which is an auto-adaptative genetic algorithm.

By clicking on *parameters*⋯ button, a dialog is opened allowing defining parameters. These are:

- ***Population size*: this is the number of configurations kept after a** generation. If it is set to 50, your final result can't contain more than 50 different points.
- ***Off spring rate*: this is the number of children per adult obtained** after combination process. If it is set to 3, each generation will contain 150 individual (considering population size is 50).
- ***Max generations*: this number defines the number of generations** after which optimization should stop. In our case, each generation corresponds to 150 simulations. Note that you can still stop optimization while it is running by clicking on *stop* button (which will appear once optimization is launched).

Therefore, you can set a really high number and still stop optimization when you want without losing results obtained until there.

- ***Save frequency*: during optimization, best configurations can be** regularly saved. It allows to analyze evolution of best configurations but also to restart an optimization from previously obtained results. A Save Frequency parameter set to 3 means that after three generations, a file is automatically created containing best configurations. These files are named iteraion1.sav, iteration2.sav and are store in *Temp* directory, and moved to *SolvedProblems* directory when optimization is finished.
- ***ReinitStdDev*: this is a specific parameter of EAAdapt1. It defines** whether standard deviation of variables should be reinitialized. It is used only if you start optimization from previously obtained configurations (using *Use start file* option). Setting it to yes (1) will, in most of cases, lead to a spread research of optimized configurations, forgetting parameters' variations' reduction obtained in previous optimization.

Use start file

As indicated before, it is possible to pursue an optimization finished or stopped. To do this, you must enable *Use start file* option and select file from which optimization should be started. This file is an *iteration_.sav* file created in previous optimization. It is stored in corresponding *SolvedProblems* folder (*iteration10.sav* corresponds to the tenth generation of previous optimization).

***Note that this functionality can only work with same variables and objectives*.** However, minimum, maximum of variables and objectives can be changed before pursuing an optimization.

Launch

You can now launch Optimization by clicking *Launch* button.

Stopping Optimization

Optimization will be stopped when the generation counter will reach the generation number defined in parameters. However, you can still stop the optimization while it is running without loosing obtained results. To do this, click on *Stop* button. Note that this will not immediately stop optimization: it will first finish the current generation.

This stop function is especially useful when optimum points do not vary any more between generations. This can be easily observed since at each generation, the optimum objectives values and corresponding parameters are displayed in log window.

11.6.3 Results

The result tab appear when the optimization is finished. It consists of two parts: a table where variables are displayed and a plot region.

Obtaining all Variable Values

During optimization, the values of optimized variables and objectives are memorized. The others are not. To get these last, you must recomputed corresponding points. To achieve this, select one or several points in point's list region and click on *recompute*.

For each point, it will simulate model setting input parameters to point corresponding values. All values of this point (including those which are not optimization parameters neither objectives).

11.6.4 Window Regions in OMOptim GUI

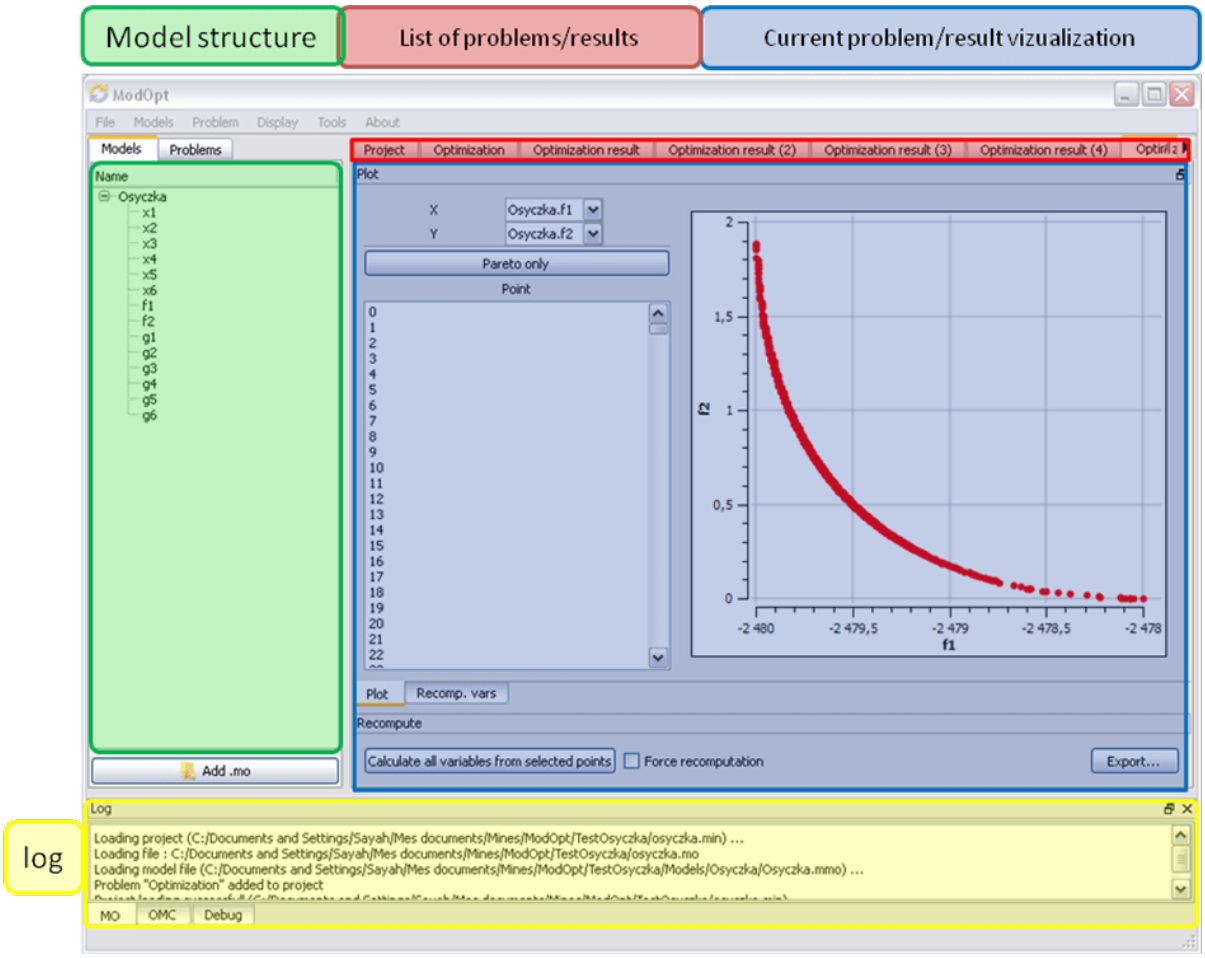


Figure 11.6: Window regions in OMOptim GUI.

第12章 Parameter Sensitivities with OpenModelica

This section describes the use of OpenModelica to compute parameter sensitivities using forward sensitivity analysis together with the Sundials/IDA solver.

Note: this is a very short preliminary description which soon will be considerably improved, since this a rather new feature and will continuous improved.

Note: OpenModelica version 1.10 or newer is required.

12.1 Background

Parameter sensitivity analysis aims at analyzing the behavior of the corresponding model states w.r.t. model parameters.

Formally, consider a Modelica model as a DAE system:

$$F(x, \dot{x}, y, p, t) = 0 \; x(t_0) = x_0(p)$$

where $x(t) \in \mathbf{R}^n$ represent state variables, $\dot{x}(t) \in \mathbf{R}^n$ represent state derivatives, $y(t) \in \mathbf{R}^k$ represent algebraic variables, $p \in \mathbf{R}^m$ model parameters.

For parameter sensitivity analysis the derivatives

$$\frac{\partial x}{\partial p}$$

are required which quantify, according to their mathematical definition, the impact of parameters p on states x. In the Sundials/IDA implementation the derivatives are used to evolve the solution over the time by:

$$\dot{s}_i = \frac{\partial x}{\partial p_i}$$

12.2 An Example

This section demonstrates the usage of the sensitivities analysis in OpenModelica on an example. This module is enabled by the following OpenModelica compiler flag:

Listing 12.1: LotkaVolterra.mo

```
model LotkaVolterra
  Real x(start=5, fixed=true),y(start=3, fixed=true);
  parameter Real mu1=5,mu2=2;
  parameter Real lambda1=3,lambda2=1;
equation
  0 = x*(mu1-lambda1*y) - der(x);
  0 = -y* (mu2 -lambda2*x) - der(y);
end LotkaVolterra;
```

Also for the simulation it is needed to set `IDA` as solver integration method and add a further simulation flag `-idaSensitivity` to calculate the parameter sensitivities during the normal simulation.

```
>>> simulate(LotkaVolterra, method="ida", simflags="-idaSensitivity")
record SimulationResult
    resultFile = "",
    simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 500,
 tolerance = 1e-06, method = 'ida', fileNamePrefix = 'LotkaVolterra', options = '',
 outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = '-
↪idaSensitivity'",
    messages = "Simulation execution failed for model: LotkaVolterra
assert             | debug   | ##IDA## set IDASensInit failed!
",
    timeFrontend = 0.0171569,
    timeBackend = 0.0186277,
    timeSimCode = 0.0017641,
    timeTemplates = 0.0136214,
    timeCompile = 2.5158786
end SimulationResult;
```

Now all calculated sensitivities are stored into the results mat file under the $Sensitivities block, where all currently every **top-level** parameter of the Real type is used to calculate the sensitivities w.r.t. **every state**.

エラー: Unable to execute gnuplot directive

Expected {quoted string, starting with " ending with " | Combine:({["-"] {"0" | W:(1234..., 0123...)} [{"." W:(0123...)}] [{W:(eE) W:(0123..., 0123...)}]}) | Forward: Group:({Suppress:("record") Suppress:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}}) Dict:(Group:({{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} Suppress:("=")} Forward: {{{{{{{{{quoted string, starting with " ending with " | Combine:({{{{["-"] {"0" | W:(1234..., 0123...)}} [{"." W:(0123...)}]} [{W:(eE) W:(0123..., 0123...)}]})} | Forward: None} | Group:({{Suppress:("{") [Forward: None [, Forward: None]...]} Suppress:("}")})} | Group:({{Suppress:("(") [Forward: None [, Forward: None]...]} Suppress:(")")})} | {{{Suppress:("SOME") Suppress:("(")} : ...} Suppress:(")")}} | "true"} | "false"} | {{"NONE" Suppress:("(")} Suppress:(")")}} | Combine:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | Group:({Suppress:("{") [Forward: None [, Forward: None]...] Suppress:("}")}) | Group:({Suppress:("(") [Forward: None [, Forward: None]...] Suppress:(")")}) | {Suppress:("SOME") Suppress:("(") Forward: {{{{{{{{{quoted string, starting with " ending with " | Combine:({{{{["-"] {"0" | W:(1234..., 0123...)}} [{"." W:(0123...)}]} [{W:(eE) W:(0123..., 0123...)}]})} | Forward: None} | Group:({{Suppress:("{") [Forward: None [, Forward: None]...]} Suppress:("}")})} | Group:({{Suppress:("(") [Forward: None [, For-

ward: None]...]} Suppress:(")")})} | {{{Suppress:("SOME") Suppress:("(")} : ...} Suppress:(")")}} | "true"} | "false"} | {{"NONE" Suppress:("(")} Suppress:(")")}} | Combine:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}}} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}})} Suppress:(")")} | "true" | "false" | {"NONE" Suppress:("(") Suppress:(")")} | Combine:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}}} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}})}, found end of text (at char 1), (line:2, col:1) Traceback (most recent call last): File "/home/.local/lib/python3.6/site-packages/pyparsing.py", line 1683, in _parseNoCache loc, tokens = self.parseImpl(instring, preloc, doActions) File "/home/.local/lib/python3.6/site-packages/pyparsing.py", line 3515, in parseImpl result = instring[loc] == self.firstQuoteChar and self.re_match(instring, loc) or None IndexError: string index out of range

During handling of the above exception, another exception occurred:

Traceback (most recent call last): File "/home/OpenModelica/doc/UsersGuide/source/sphinxcontribopenmodelica.py", line 173, in run filename = os.path.abspath(self.options.get('filename') or omc.sendExpression("currentSimulationResult")) File "/home/.local/lib/python3.6/site-packages/OMPython/__init__.py", line 606, in sendExpression answer = OMTypedParser.parseString(result) File "/home/.local/lib/python3.6/site-packages/OMPython/OMTypedParser.py", line 120, in parseString return omcGrammar.parseString(string)[0] File "/home/.local/lib/python3.6/site-packages/pyparsing.py", line 1955, in parseString raise exc File "/home/.local/lib/python3.6/site-packages/pyparsing.py", line 1685, in _parseNoCache raise ParseException(instring, len(instring), self.errmsg, self) pyparsing.ParseException: Expected {quoted string, starting with " ending with " | Combine:({["-"] {"0" | W:(1234..., 0123...)} [{"." W:(0123...)}] [{W:(eE) W:(0123..., 0123...)}]}) | Forward: Group:({Suppress:("record") Suppress:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}}) Dict:(Group:({{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} Suppress:("=")} Forward: {{{{{{{{{quoted string, starting with " ending with " | Combine:({{{{["-"] {"0" | W:(1234..., 0123...)}} [{"." W:(0123...)}]} [{W:(eE) W:(0123..., 0123...)}]})} | Forward: None} | Group:({{Suppress:("{") [Forward: None [, Forward: None]...]} Suppress:("}")})} | Group:({{Suppress:("(") [Forward: None [, Forward: None]...]} Suppress:(")")})} | {{{Suppress:("SOME") Suppress:("(")} : ...} Suppress:(")")}} | "true"} | "false"} | {{"NONE" Suppress:("(")} Suppress:(")")}} | Combine:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | Group:({Suppress:("{") [Forward: None [, Forward: None]...] Suppress:("}")}) | Group:({Suppress:("(") [Forward: None [, Forward: None]...] Suppress:(")")}) | {Suppress:("SOME") Suppress:("(") Forward: {{{{{{{{{quoted string, starting with " ending with " | Combine:({{{{["-"] {"0" | W:(1234..., 0123...)}} [{"." W:(0123...)}]} [{W:(eE) W:(0123..., 0123...)}]})} | Forward: None} | Group:({{Suppress:("{") [Forward: None [, Forward: None]...]} Suppress:("}")})} | Group:({{Suppress:("(") [Forward: None [, Forward: None]...]} Suppress:(")")})} | {{{Suppress:("SOME") Suppress:("(")} : ...} Suppress:(")")}} | "true"} | "false"} | {{"NONE" Suppress:("(")} Suppress:(")")}} | Combine:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}}} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}})} Suppress:(")")} | "true" |

"false" | {"NONE" Suppress:("(") Suppress:(")")} | Combine:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}}} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}})}, found end of text (at char 1), (line:2, col:1)

エラー: Unable to execute gnuplot directive

Expected {quoted string, starting with " ending with " | Combine:({["-"] {"0" | W:(1234..., 0123...)} [{"." W:(0123...)}] [{W:(eE) W:(0123..., 0123...)}]}) | Forward: Group:({Suppress:("record") Suppress:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}}) Dict:(Group:({{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} Suppress:("=")} Forward: {{{{{{{{{quoted string, starting with " ending with " | Combine:({{{{["-"] {"0" | W:(1234..., 0123...)}} [{"." W:(0123...)}]} [{W:(eE) W:(0123..., 0123...)}]})} | Forward: None} | Group:({{Suppress:("{") [Forward: None [, Forward: None]...]} Suppress:("}")})} | Group:({{Suppress:("(") [Forward: None [, Forward: None]...]} Suppress:(")")})} | {{{Suppress:("SOME") Suppress:("(")} : ...} Suppress:(")")}} | "true"} | "false"} | {{"NONE" Suppress:("(")} Suppress:(")")}} | Combine:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | Group:({Suppress:("{") [Forward: None [, Forward: None]...] Suppress:("}")}) | Group:({Suppress:("(") [Forward: None [, Forward: None]...] Suppress:(")")}) | {Suppress:("SOME") Suppress:("(") Forward: {{{{{{{{{quoted string, starting with " ending with " | Combine:({{{{["-"] {"0" | W:(1234..., 0123...)}} [{"." W:(0123...)}]} [{W:(eE) W:(0123..., 0123...)}]})} | Forward: None} | Group:({{Suppress:("{") [Forward: None [, Forward: None]...]} Suppress:("}")})} | Group:({{Suppress:("(") [Forward: None [, Forward: None]...]} Suppress:(")")})} | {{{Suppress:("SOME") Suppress:("(")} : ...} Suppress:(")")}} | "true"} | "false"} | {{"NONE" Suppress:("(")} Suppress:(")")}} | Combine:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}}} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}})} Suppress:(")")} | "true" | "false" | {"NONE" Suppress:("(") Suppress:(")")} | Combine:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}}} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}})}, found end of text (at char 1), (line:2, col:1) Traceback (most recent call last): File "/home/.local/lib/python3.6/site-packages/pyparsing.py", line 1683, in _parseNoCache loc, tokens = self.parseImpl(instring, preloc, doActions) File "/home/.local/lib/python3.6/site-packages/pyparsing.py", line 3515, in parseImpl result = instring[loc] == self.firstQuoteChar and self.re_match(instring, loc) or None IndexError: string index out of range

During handling of the above exception, another exception occurred:

Traceback (most recent call last): File "/home/OpenModelica/doc/UsersGuide/source/sphinxcontribopenmodelica.py", line 173, in run filename = os.path.abspath(self.options.get('filename') or omc.sendExpression("currentSimulationResult")) File "/home/.local/lib/python3.6/site-packages/OMPython/__init__.py", line 606, in sendExpression answer = OMTypedParser.parseString(result)

File "/home/.local/lib/python3.6/site-packages/OMPython/OMTypedParser.py", line 120, in parseString return omcGrammar.parseString(string)[0] File "/home/.local/lib/python3.6/site-packages/pyparsing.py", line 1955, in parseString raise exc File "/home/.local/lib/python3.6/site-packages/pyparsing.py", line 1685, in _parseNoCache raise ParseException(instring, len(instring), self.errmsg, self) pyparsing.ParseException: Expected {quoted string, starting with " ending with " | Combine:({["-"] {"0" | W:(1234..., 0123...)} [{"." W:(0123...)}] [{W:(eE) W:(0123..., 0123...)}]}) | Forward: Group:({Suppress:("record") Suppress:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}}) Dict:(Group:({{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} Suppress:("=")} Forward: {{{{{{{{{quoted string, starting with " ending with " | Combine:({{{{"-"] {"0" | W:(1234..., 0123...)}} [{"." W:(0123...)}]} [{W:(eE) W:(0123..., 0123...)}]})} | Forward: None} | Group:({{Suppress:("{") [Forward: None [, Forward: None]...]} Suppress:("}")})} | Group:({{Suppress:("(") [Forward: None [, Forward: None]...]} Suppress:(")")})} | {{{Suppress:("SOME") Suppress:("(")} : ...} Suppress:(")")}} | "true"} | "false"} | {{"NONE" Suppress:("(")} Suppress:(")")}} | Combine:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | Group:({Suppress:("{") [Forward: None [, Forward: None]...] Suppress:("}")}) | Group:({Suppress:("(") [Forward: None [, Forward: None]...] Suppress:(")")}) | {Suppress:("SOME") Suppress:("(") Forward: {{{{{{{{{quoted string, starting with " ending with " | Combine:({{{{"-"] {"0" | W:(1234..., 0123...)}} [{"." W:(0123...)}]} [{W:(eE) W:(0123..., 0123...)}]})} | Forward: None} | Group:({{Suppress:("{") [Forward: None [, Forward: None]...]} Suppress:("}")})} | Group:({{Suppress:("(") [Forward: None [, Forward: None]...]} Suppress:(")")})} | {{{Suppress:("SOME") Suppress:("(")} : ...} Suppress:(")")}} | "true"} | "false"} | {{"NONE" Suppress:("(")} Suppress:(")")}} | Combine:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}}} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}})} Suppress:(")")} | "true" | "false" | {"NONE" Suppress:("(") Suppress:(")")} | Combine:(Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} Forward: {{{{W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '} "."} : ...} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}}} | {W:(ABCD..., ABCD...) | quoted string, starting with ' ending with '}})}, found end of text (at char 1), (line:2, col:1)

第13章　PDEModelica1

PDEModelica1 is nonstandardised experimental Modelica language extension for 1-dimensional partial differential extensions (PDE).

It is enabled using compiler flag `--grammar=PDEModelica`. Compiler flags may be set e.g. in OMEdit (Tools->Options->Simulation->OMC Flags) or in the OpenModelica script using command

```
```

13.1 PDEModelica1 language elements

Let us introduce new PDEModelica1 language elements by an advection equation example model:

```
model Advection "advection equation"
  parameter Real pi = Modelica.Constants.pi;
  parameter DomainLineSegment1D omega(L = 1, N = 100)  "domain";
  field Real u(domain = omega)                         "field";
initial equation
  u = sin(2*pi*omega.x)                                "IC";
equation
  der(u) + pder(u,x) = 0   indomain omega              "PDE";
  u = 0                    indomain omega.left         "BC";
  u = extrapolateField(u)  indomain omega.right        "extrapolation";
end Advection;
```

エラー:

[<interactive>:4:14-4:14:writable] Error: Missing token: SEMICOLON

The domain `omega` represents the geometrical domain where the PDE holds. The domain is defined using the built-in record `DomainLineSegment1D`. This record contains among others `L` – the length of the domain, `N` – the number of grid points, `x` – the coordinate variable and the regions `left`, `right` and `interior`, representing the left and right boundaries and the interior of the domain.

The field variable `u` is defined using a new keyword `field`. The `domain` is a mandatory attribute to specify the domain of the field.

The `indomain` operator specifies where the equation containing the field variable holds. It is utilised in the initial conditions (IC) of the fields, in the PDE and in the boundary conditions (BC). The syntax is

```
anEquation indomain aDomain.aRegion;
```

If the region is omitted, `interior` is the default (e.g. the PDE in the example above).

The IC of the field variable u is written using an expression containing the coordinate variable `omega.x`.

The PDE contains a partial space derivative written using the `pder` operator. Also the second derivative is allowed (not in this example), the syntax is e.g. `pder(u,x,x)`. It is not necessary to specify the domain of coordinate in pder (to write e.g. `pder(u,omega.x)`, even though `x` is a member of `omega`.

13.2 Limitations

BCs may be written only in terms of variables that are spatially differentiated currently.

All fields that are spatially differentiated must have either BC or extrapolation at each boundary. This extrapolation should be done automatically by the compiler, but this has not been implemented yet. The current workaround is the usage of the `extrapolateField()` operator directly in the model.

If-equations are not spported yet, if-expressions must be used instead.

13.3 Viewing results

During translation field variables are replaced with arrays. These arrays may be plotted using 配列のプロット or even better using アレイパラメトリックプロット (to plot x-coordinate versus a field).

第14章　MDT – The OpenModelica Development Tooling Eclipse Plugin

14.1 Introduction

The Modelica Development Tooling (MDT) Eclipse Plugin as part of OMDev – The OpenModelica Development Environment integrates the OpenModelica compiler with Eclipse. MDT, together with the OpenModelica compiler, provides an environment for working with Modelica and MetaModelica development projects. This plugin is primarily intended for tool developers rather than application Modelica modelers.

The following features are available:

- Browsing support for Modelica projects, packages, and classes
- Wizards for creating Modelica projects, packages, and classes
- Syntax color highlighting
- Syntax checking
- Browsing of the Modelica Standard Library or other libraries
- Code completion for class names and function argument lists
- Goto definition for classes, types, and functions
- Displaying type information when hovering the mouse over an identifier.

14.2 Installation

The installation of MDT is accomplished by following the below installation instructions. These instructions assume that you have successfully downloaded and installed Eclipse (http://www.eclipse.org).

The latest installation instructions are available through the OpenModelica Trac.

1. Start Eclipse
2. Select Help->Software Updates->Find and Install**...** from the menu
3. Select ‘Search for new features to install’ and click ‘Next’
4. Select ‘New Remote Site...’
5. Enter ‘MDT’ as name and http://www.ida.liu.se/labs/pelab/modelica/OpenModelica/MDT as URL and click ‘OK’

6. Make sure ‘MDT’ is selected and click ‘Finish’
7. In the updates dialog select the ‘MDT’ feature and click ‘Next’
8. Read through the license agreement, select ‘I accept...’ and click ‘Next’
9. Click ‘Finish’ to install MDT

14.3 Getting Started

14.3.1 Configuring the OpenModelica Compiler

MDT needs to be able to locate the binary of the compiler. It uses the environment variable OPENMODELICAHOME to do so.

If you have problems using MDT, make sure that OPENMODELICAHOME is pointing to the folder where the OpenModelica Compiler is installed. In other words, OPENMODELICAHOME must point to the folder that contains the Open Modelica Compiler (OMC) binary. On the Windows platform it’s called omc.exe and on Unix platforms it’s called omc.

14.3.2 Using the Modelica Perspective

The most convenient way to work with Modelica projects is to use to the Modelica perspective. To switch to the Modelica perspective, choose the Window menu item, pick Open Perspective followed by Other**...** Select the Modelica option from the dialog presented and click OK..

14.3.3 Selecting a Workspace Folder

Eclipse stores your projects in a folder called a workspace. You need to choose a workspace folder for this session, see Figure 14.1.

14.3.4 Creating one or more Modelica Projects

To start a new project, use the New Modelica Project Wizard. It is accessible through File->New-> Modelica Project or by right-clicking in the Modelica Projects view and selecting New->Modelica Project.

You need to disable automatic build for the project(s) (Figure 14.3).

Repeat the procedure for all the projects you need, e.g. for the exercises described in the MetaModelica users guide: 01_experiment, 02a_exp1, 02b_exp2, 03_assignment, 04a_assigntwotype, etc.

NOTE: Leave open only the projects you are working on! Close all the others!

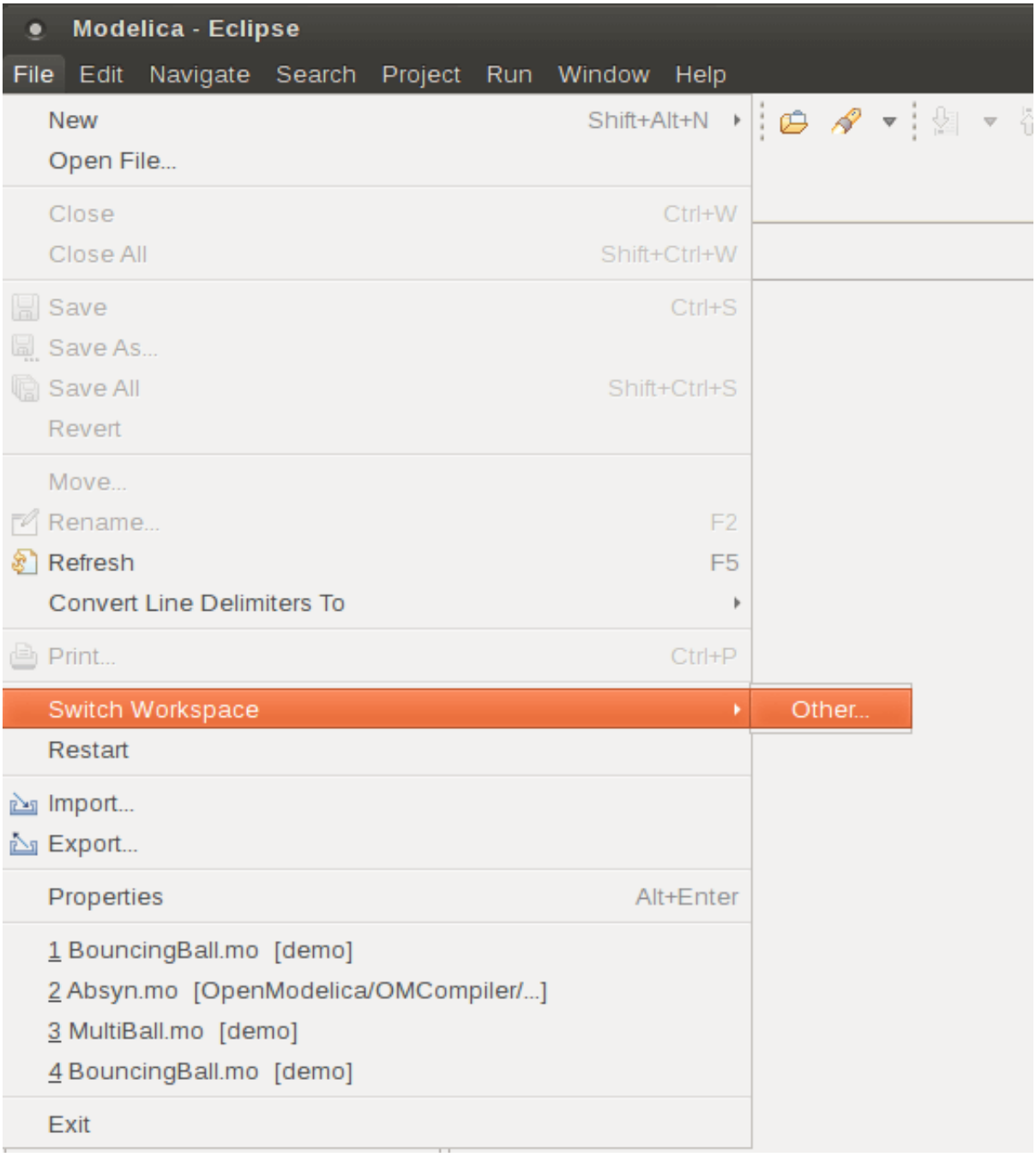


Figure 14.1: Eclipse Setup – Switching Workspace.

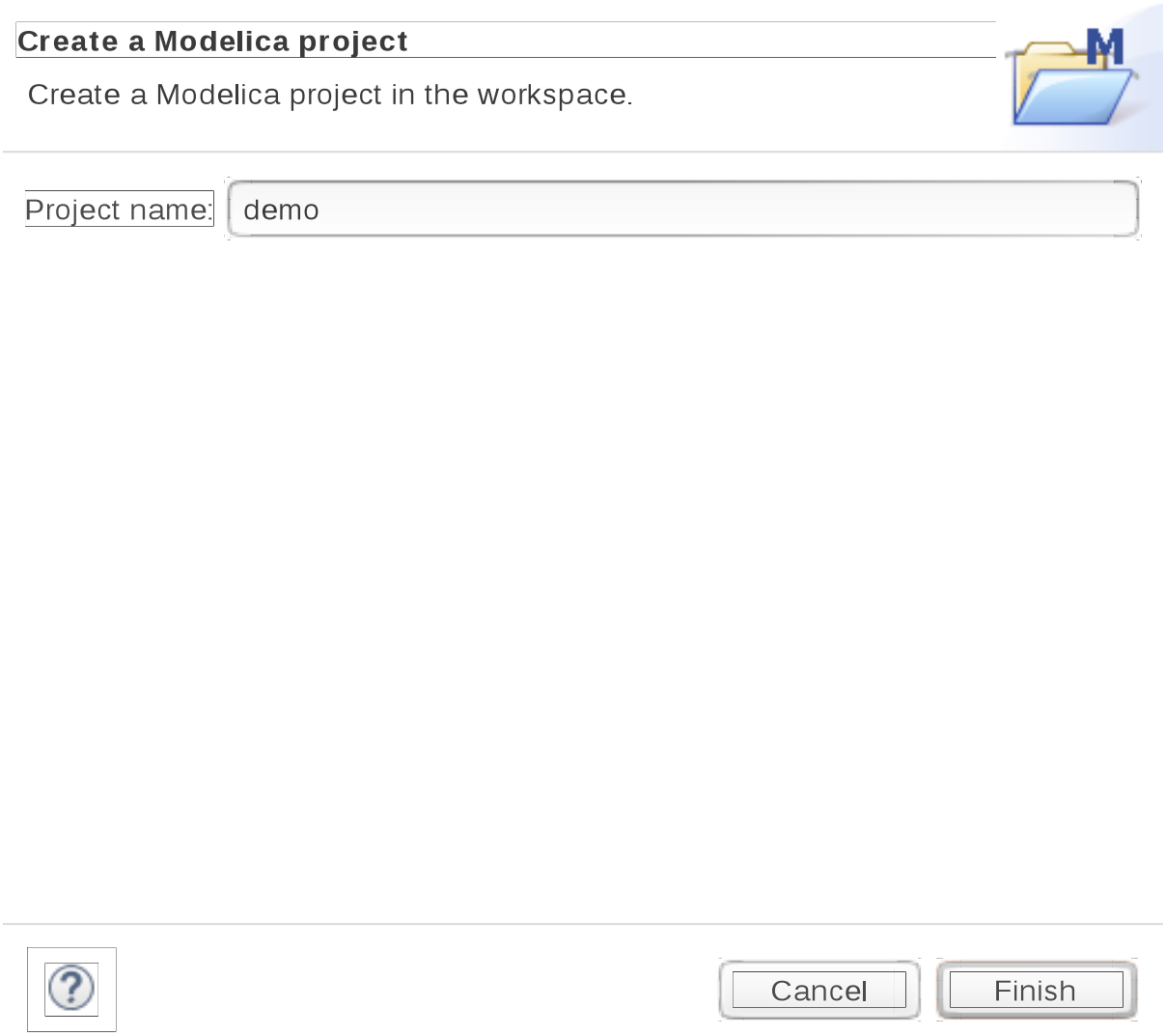


Figure 14.2: Eclipse Setup – creating a Modelica project in the workspace.

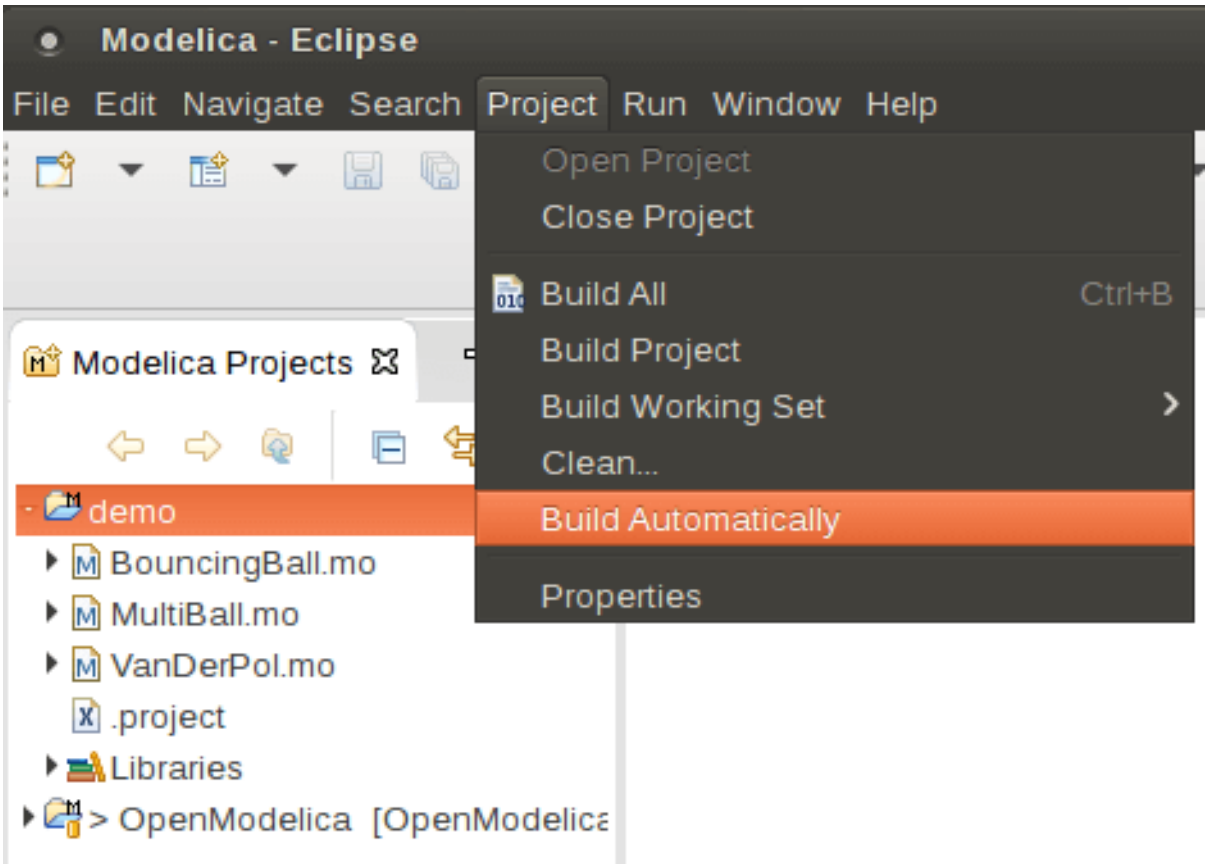


Figure 14.3: Eclipse Setup – disable automatic build for the projects.

14.3.5 Building and Running a Project

After having created a project, you eventually need to build the project (Figure 14.4).

The build options are the same as the make targets: you can build, build from scratch (clean), or run simulations depending on how the project is setup. See Figure 14.5 for an example of how omc can be compiled (`make omc` builds OMC).

14.3.6 Switching to Another Perspective

If you need, you can (temporarily) switch to another perspective, e.g. to the Java perspective for working with an OpenModelica Java client as in Figure 14.7.

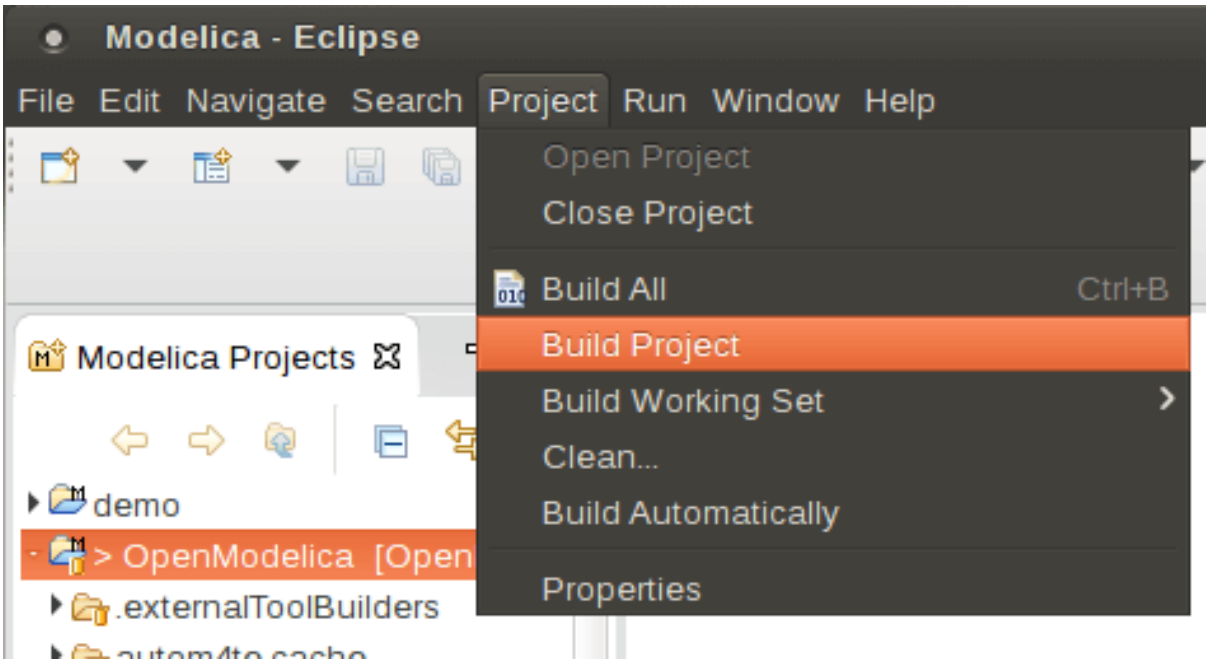


Figure 14.4: Eclipse MDT – Building a project.

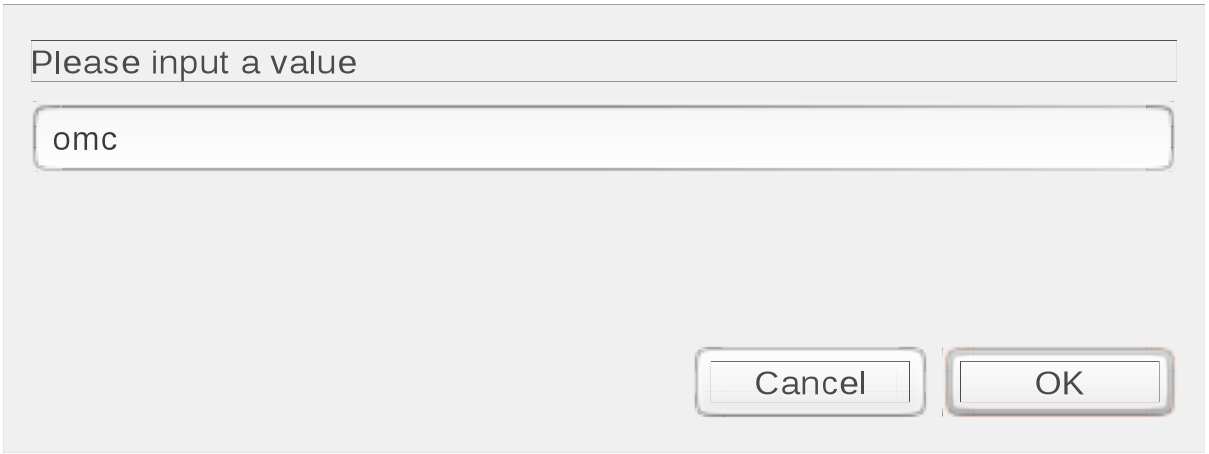


Figure 14.5: Eclipse – building a project.

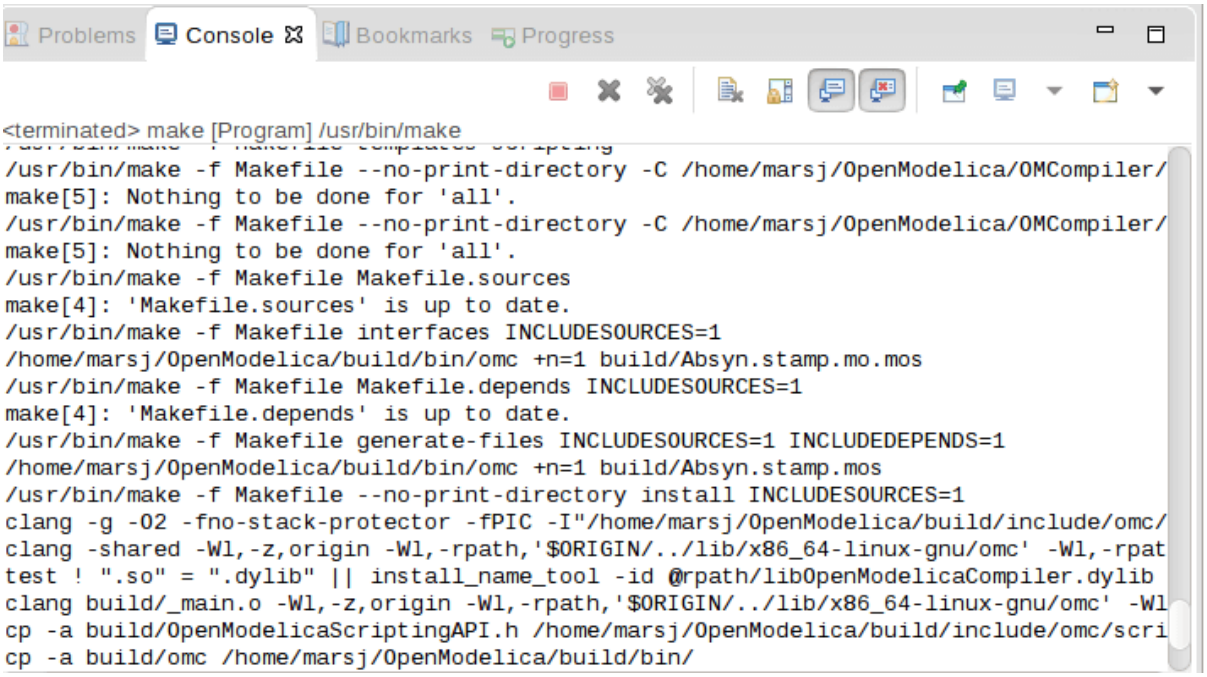


Figure 14.6: Eclipse – building a project, resulting log.

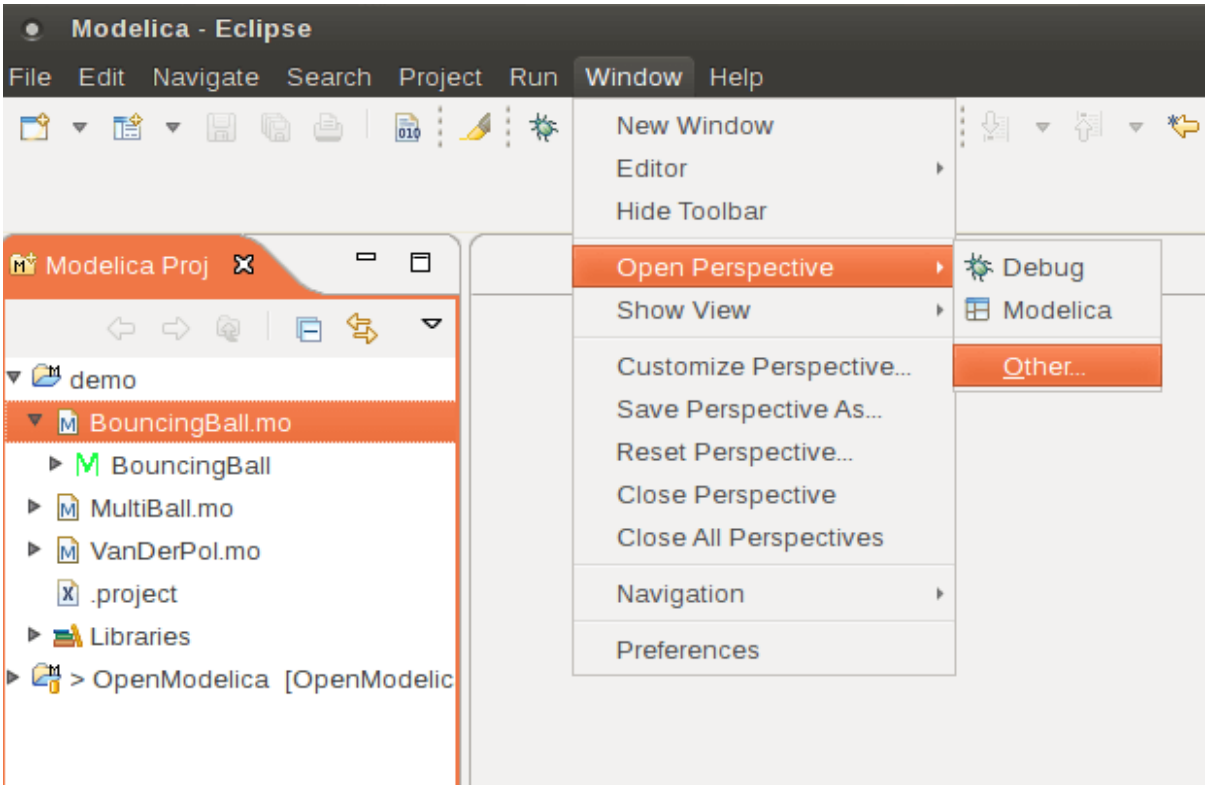


Figure 14.7: Eclipse – Switching to another perspective – e.g. the Java Perspective.

14.3.7 Creating a Package

To create a new package inside a Modelica project, select File->New->Modelica Package**.** Enter the desired name of the package and a description of what it contains. Note: for the exercises we already have existing packages.

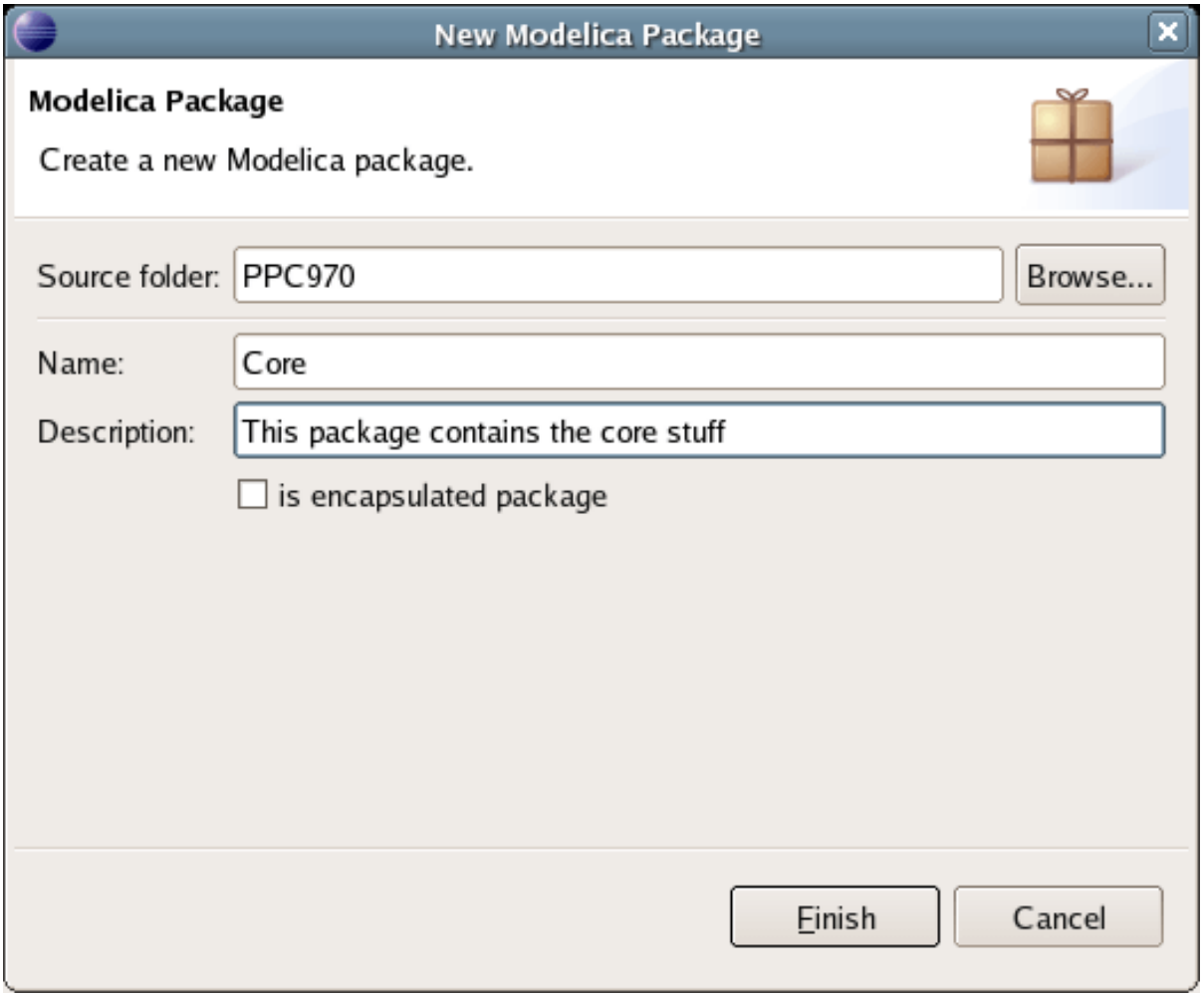


Figure 14.8: Creating a new Modelica package.

14.3.8 Creating a Class

To create a new Modelica class, select where in the hierarchy that you want to add your new class and select File->New->Modelica Class. When creating a Modelica class you can add different restrictions on what the class can contain. These can for example be model, connector, block, record, or function. When you have selected your desired class type, you can select modifiers that add code blocks to the generated code. 'Include initial code block' will for example add the line 'initial equation' to the class.

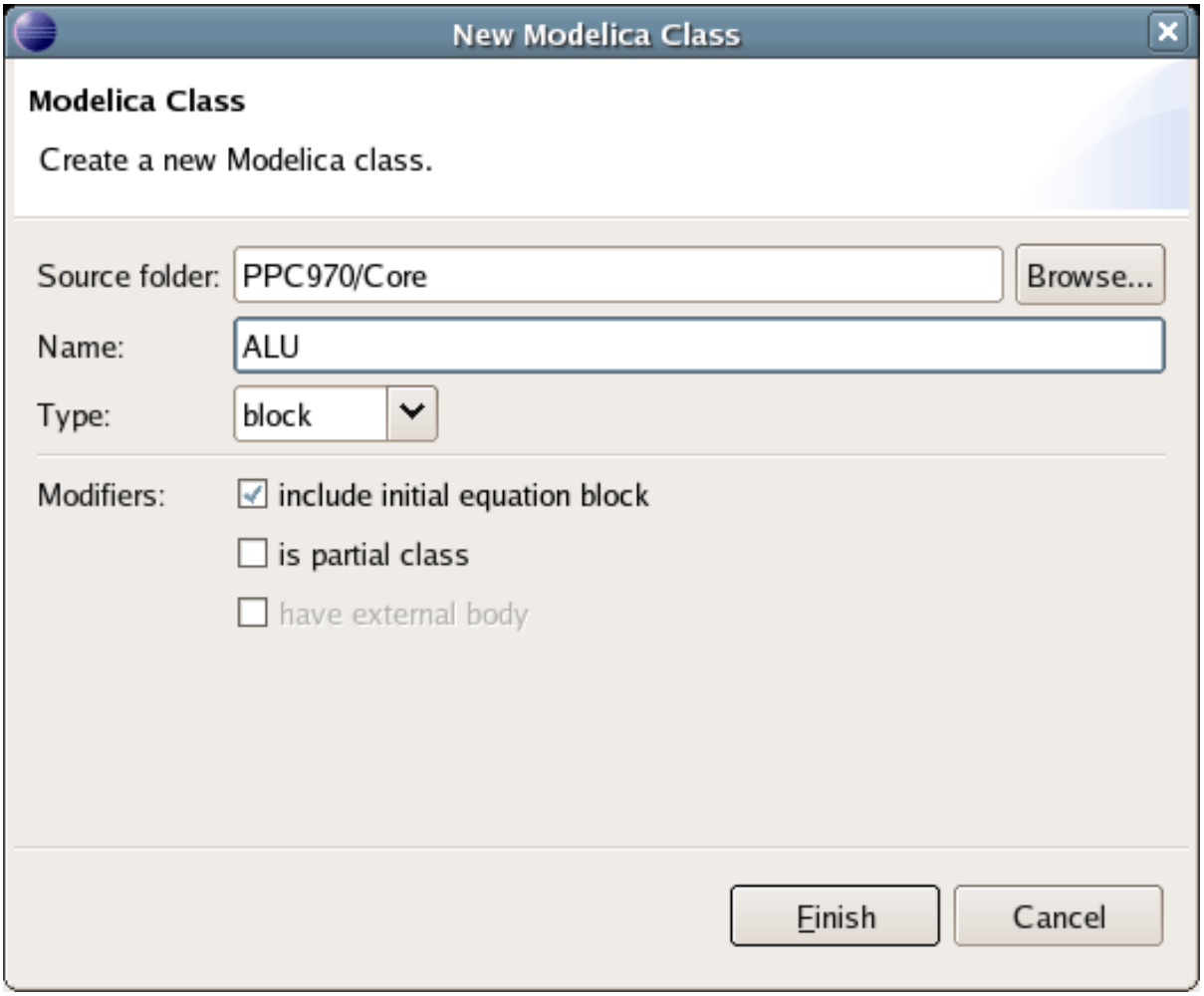


Figure 14.9: Creating a new Modelica class.

14.3.9 Syntax Checking

Whenever a build command is given to the MDT environment, modified and saved Modelica (.mo) files are checked for syntactical errors. Any errors that are found are added to the Problems view and also marked in the source code editor. Errors are marked in the editor as a red circle with a white cross, a squiggly red line under the problematic construct, and as a red marker in the right-hand side of the editor. If you want to reach the problem, you can either click the item in the Problems view or select the red box in the right-hand side of the editor.

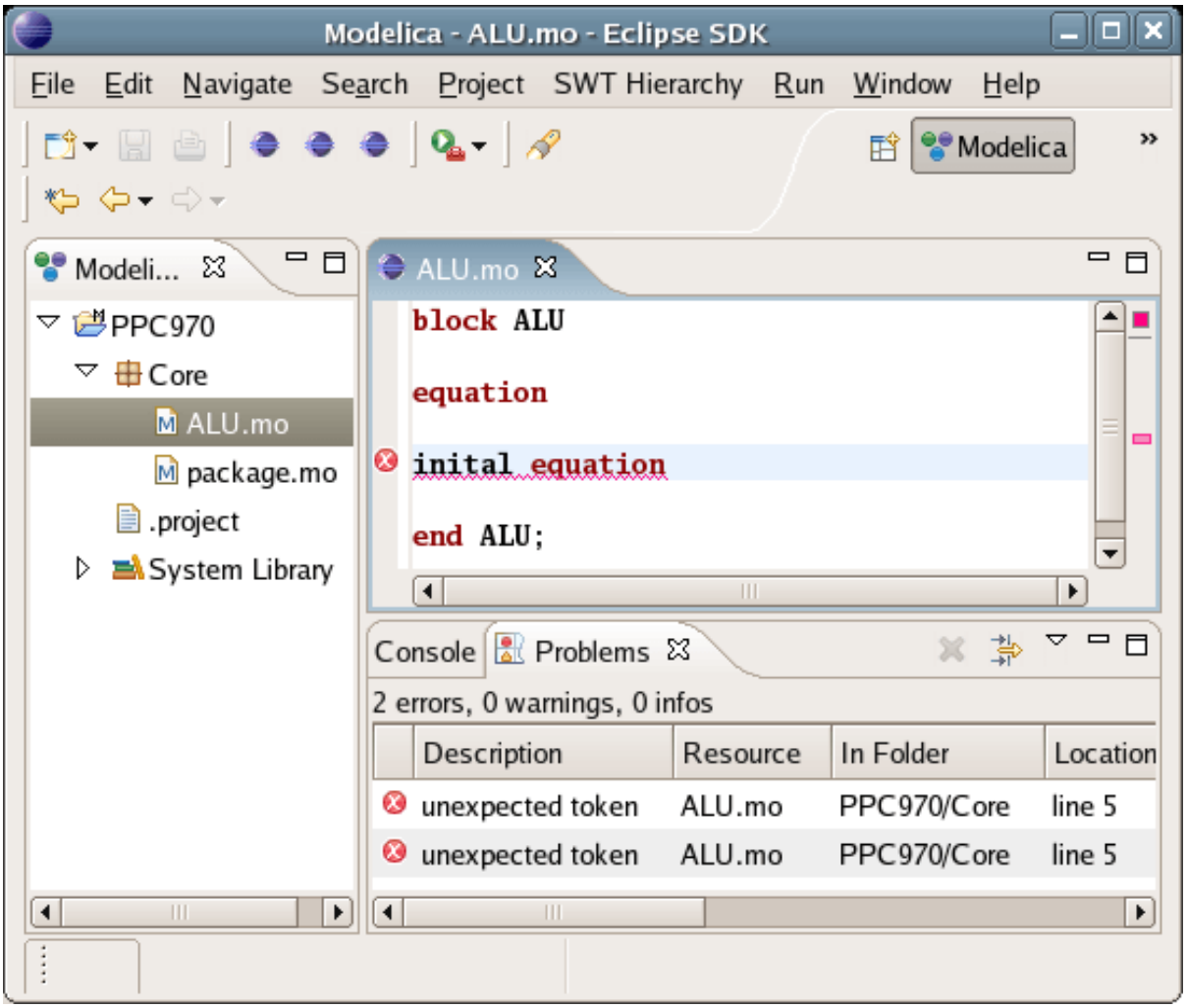


Figure 14.10: Syntax checking.

14.3.10 Automatic Indentation Support

MDT currently has support for automatic indentation. When typing the Return (Enter) key, the next line is indented correctly. You can also correct indentation of the current line or a range selection using CTRL+I or "Correct Indentation" action on the toolbar or in the Edit menu.

14.3.11 Code Completion

MDT supports Code Completion in two variants. The first variant, code completion when typing a dot after a class (package) name, shows alternatives in a menu. Besides the alternatives, Modelica documentation from comments is shown if is available. This makes the selection easyer.

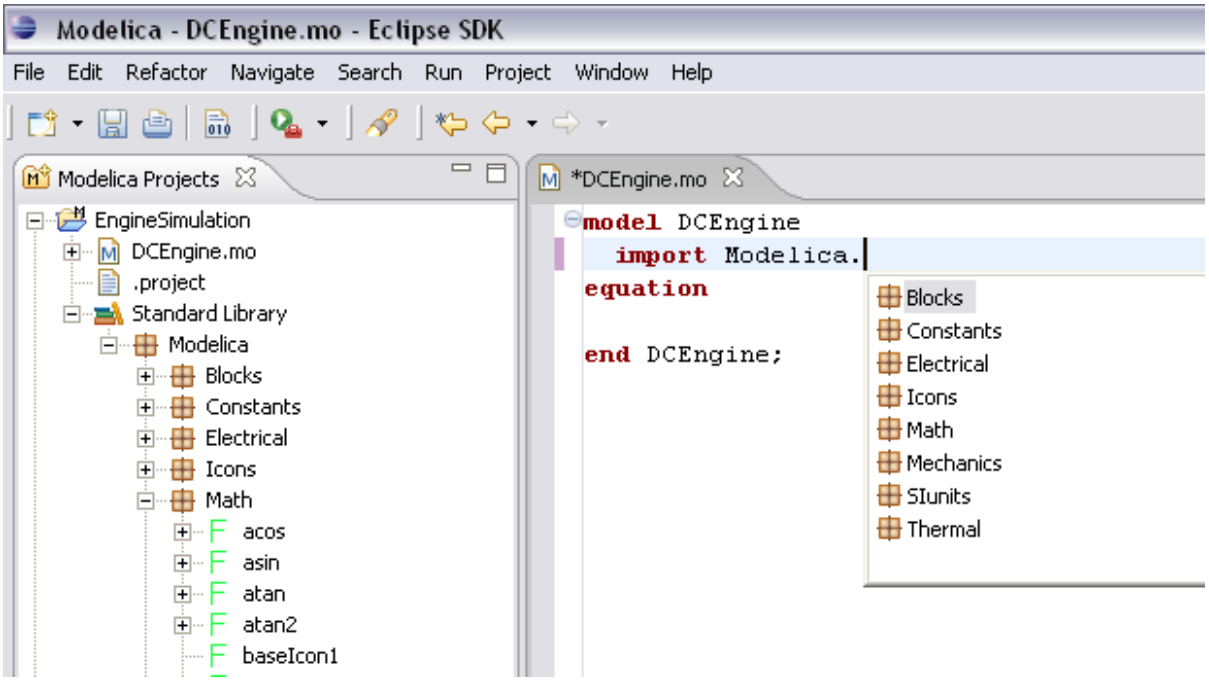


Figure 14.11: Code completion when typing a dot.

The second variant is useful when typing a call to a function. It shows the function signature (formal parameter names and types) in a popup when typing the parenthesis after the function name, here the signature Real sin(SI.Angle u) of the sin function:

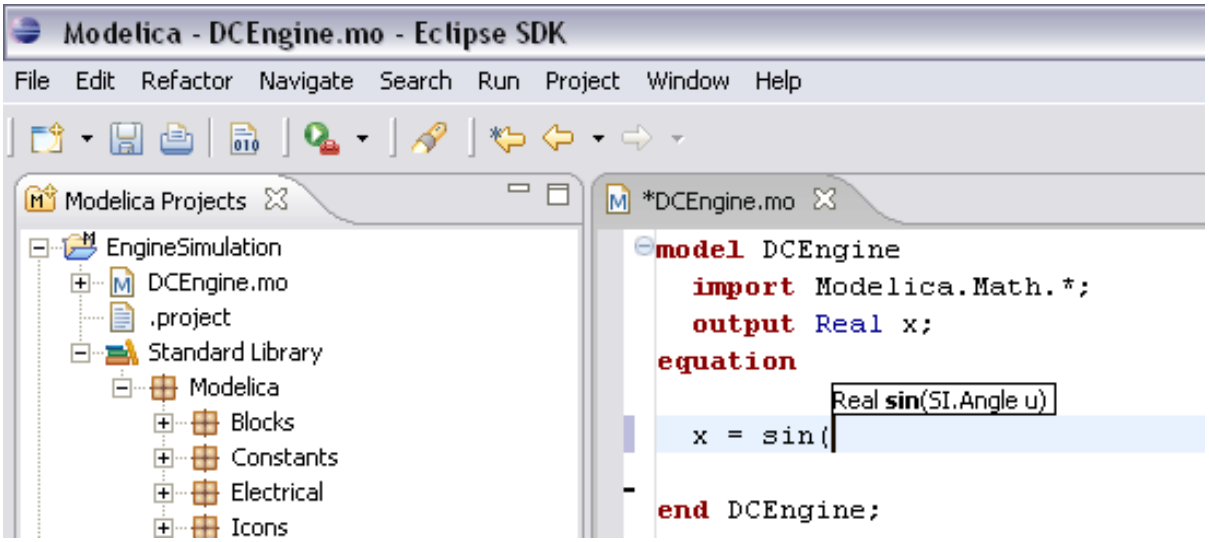


Figure 14.12: Code completion at a function call when typing left parenthesis.

14.3.12 Code Assistance on Identifiers when Hovering

When hovering with the mouse over an identifier a popup with information about the identifier is displayed. If the text is too long, the user can press F2 to focus the popup dialog and scroll up and down to examine all the text. As one can see the information in the popup dialog is syntax-highlighted.

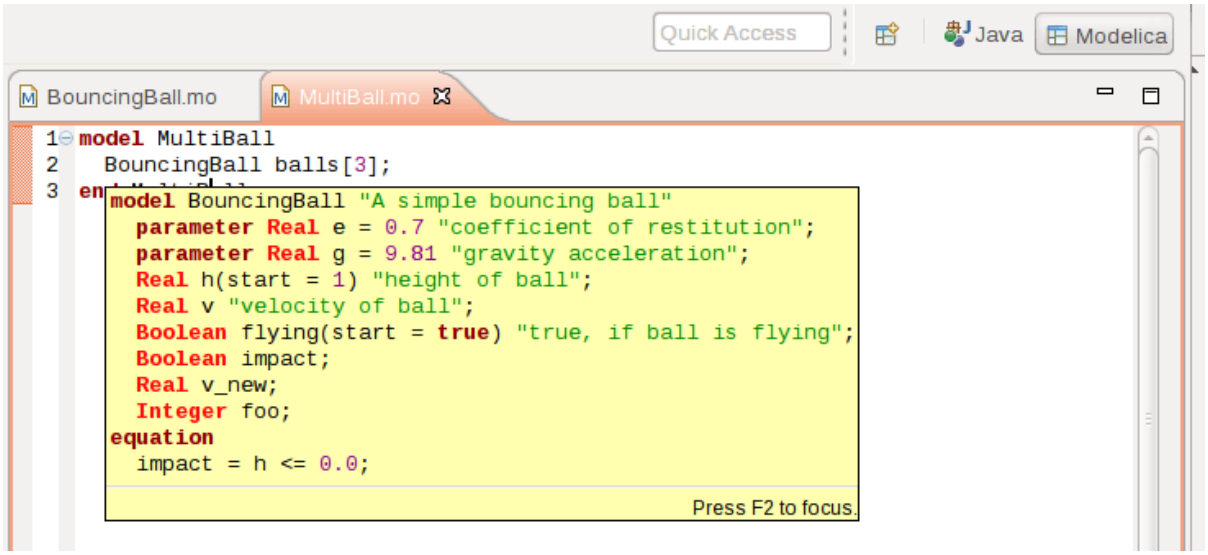


Figure 14.13: Displaying information for identifiers on hovering.

14.3.13 Go to Definition Support

Besides hovering information the user can press CTRL+click to go to the definition of the identifier. When pressing CTRL the identifier will be presented as a link and when pressing mouse click the editor will go to the definition of the identifier.

14.3.14 Code Assistance on Writing Records

When writing records, the same functionality as for function calls is used. This is useful especially in MetaModelica when writing cases in match constructs.

14.3.15 Using the MDT Console for Plotting

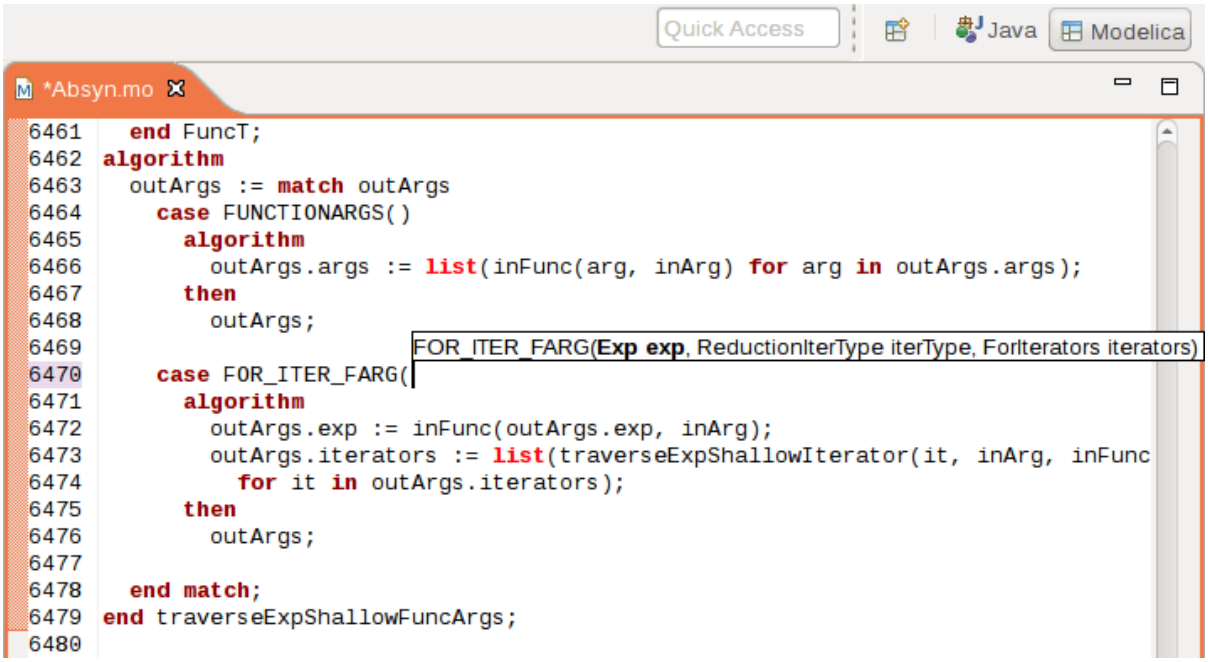


Figure 14.14: Code assistance when writing cases with records in MetaModelica.

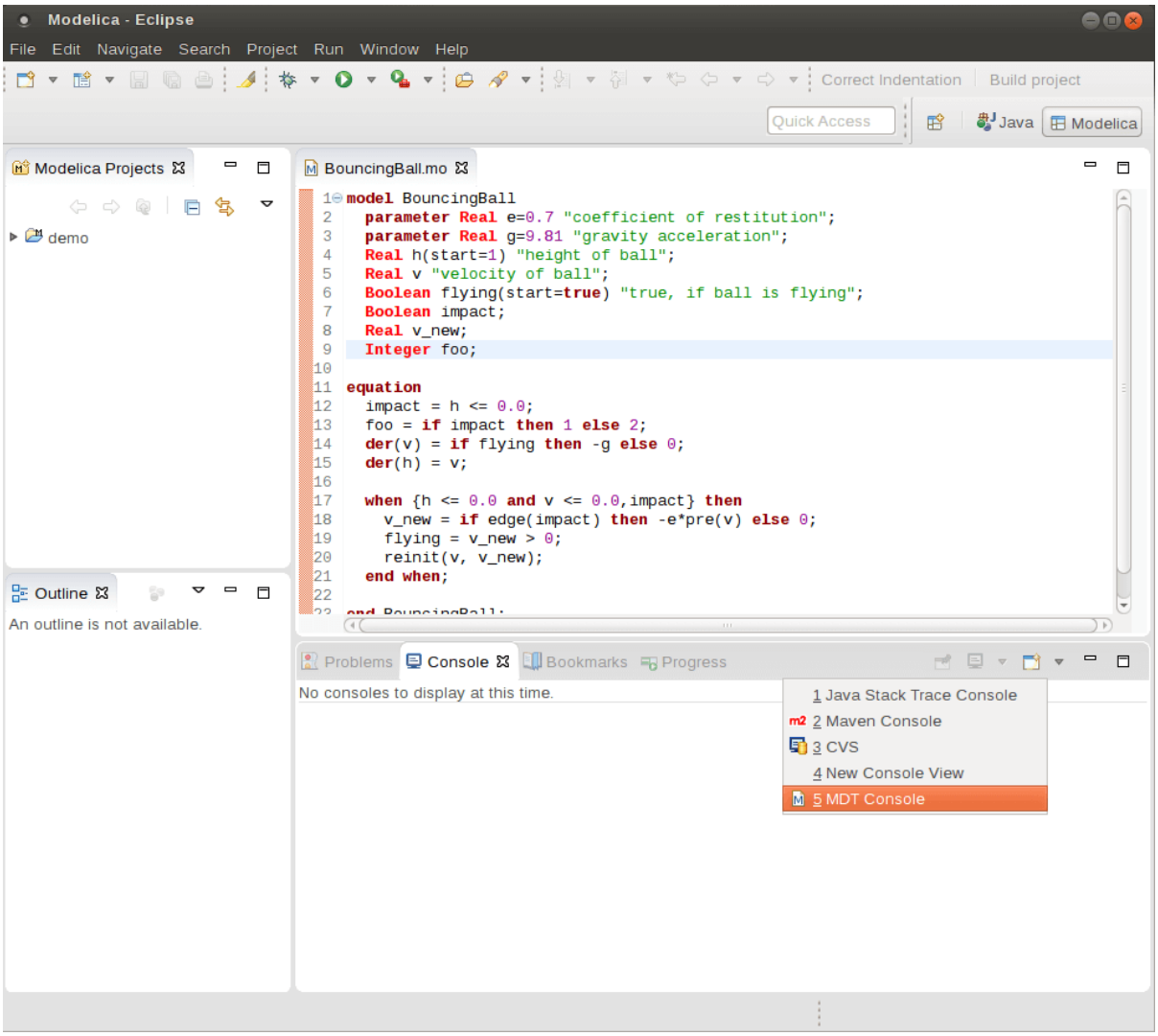


Figure 14.15: Activate the MDT Console.

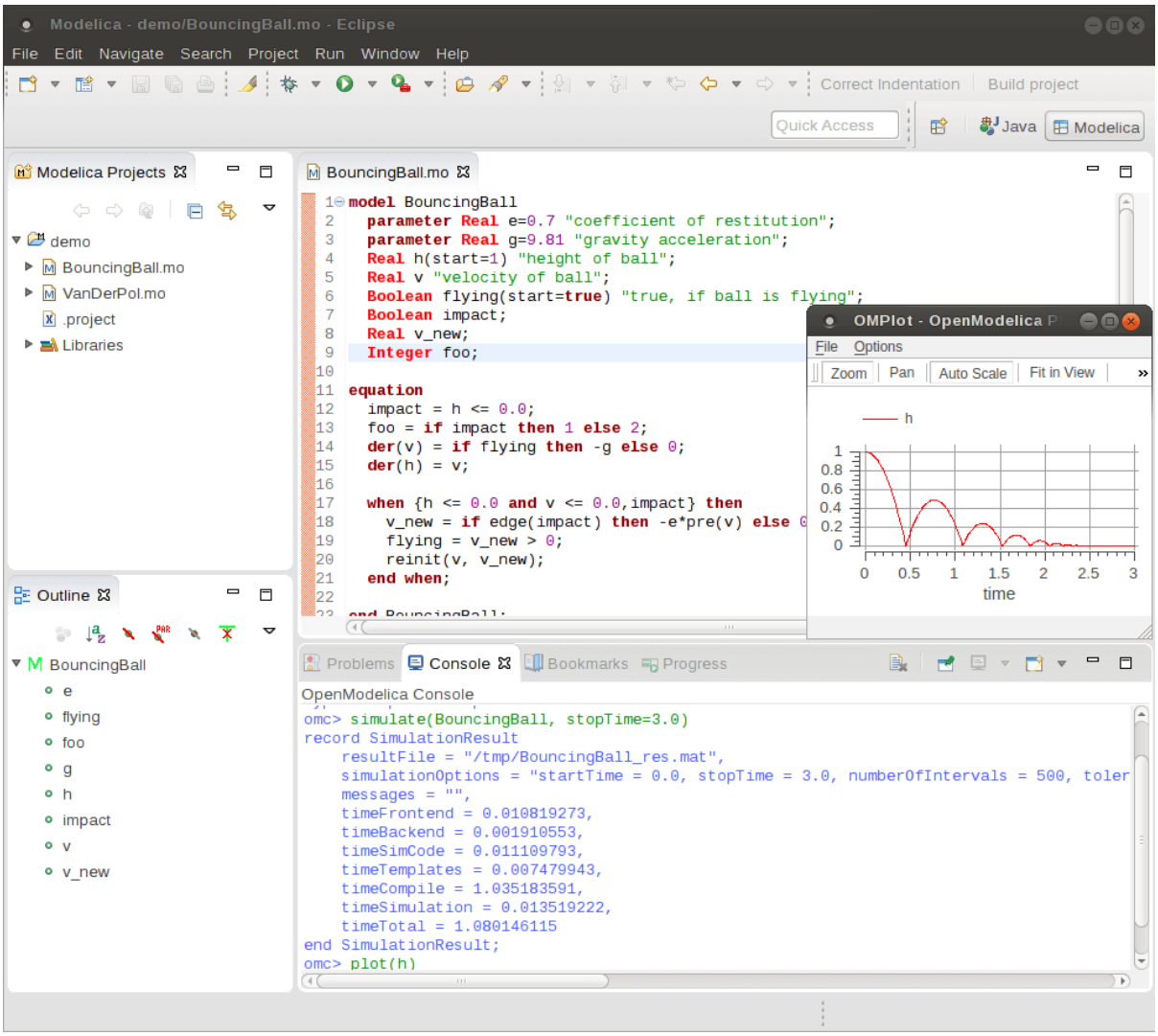


Figure 14.16: Simulation from MDT Console.

第15章　MDT Debugger for Algorithmic Modelica

The algorithmic code debugger, used for the algorithmic subset of the Modelica language as well as the MetaModelica language is described in Section *The Eclipse-based Debugger for Algorithmic Modelica*. Using this debugger replaces debugging of algorithmic code by primitive means such as print statements or asserts which is complex, time-consuming and error- prone. The usual debugging functionality found in debuggers for procedural or traditional object-oriented languages is supported, such as setting and removing breakpoints, stepping, inspecting variables, etc. The debugger is integrated with Eclipse.

15.1 The Eclipse-based Debugger for Algorithmic Modelica

The debugging framework for the algorithmic subset of Modelica and MetaModelica is based on the Eclipse environment and is implemented as a set of plugins which are available from Modelica Development Tooling (MDT) environment. Some of the debugger functionality is presented below. In the right part a variable value is explored. In the top-left part the stack trace is presented. In the middle-left part the execution point is presented.

The debugger provides the following general functionalities:

- Adding/Removing breakpoints.
- Step Over – moves to the next line, skipping the function calls.
- Step In – takes the user into the function call.
- **Step Return – complete the execution of the function and takes the** user back to the point from where the function is called.
- Suspend – interrupts the running program.

15.1.1 Starting the Modelica Debugging Perspective

To be able to run in debug mode, one has to go through the following steps:

- create a mos file
- setting the debug configuration
- setting breakpoints
- running the debug configuration

All these steps are presented below using images.

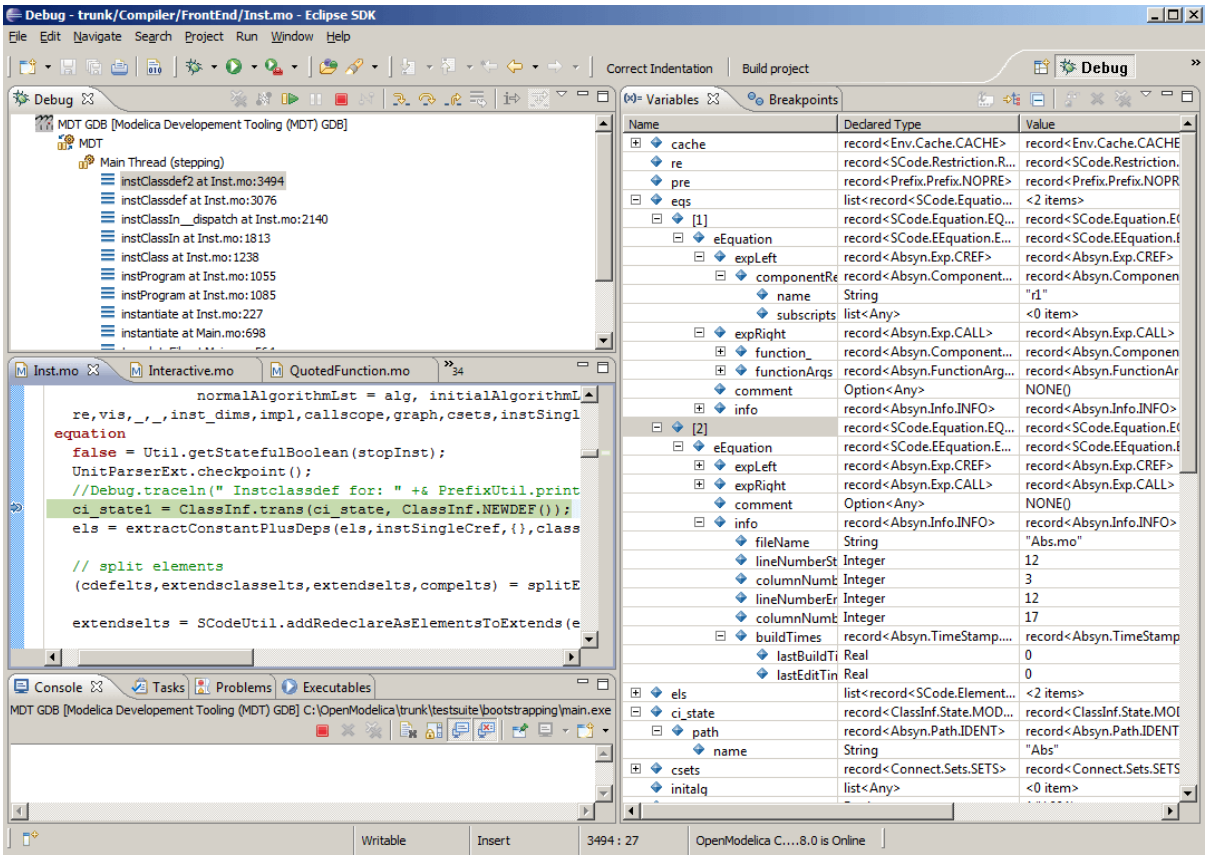


Figure 15.1: Debugging functionality.

Create mos file

In order to debug Modelica code we need to load the Modelica files into the OpenModelica Compiler. For this we can write a small script file like this:

```
function HelloWorld
  input Real r;
  output Real o;
algorithm
  o := 2 * r;
end HelloWorld;
```

```
>>> setCommandLineOptions({"-d=rml,noevalfunc","-g=MetaModelica"})
{true,true}
>>> setCFlags(getCFlags() + " -g")
true
>>> HelloWorld(120.0)
```

So lets say that we want to debug HelloWorld.mo. For that we must load it into the compiler using the script file. Put all the Modelica files there in the script file to be loaded. We should also initiate the debugger by calling the starting function, in the above code `HelloWorld(120.0);`

Setting the debug configuration

While the Modelica perspective is activated the user should click on the bug icon on the toolbar and select Debug in order to access the dialog for building debug configurations.

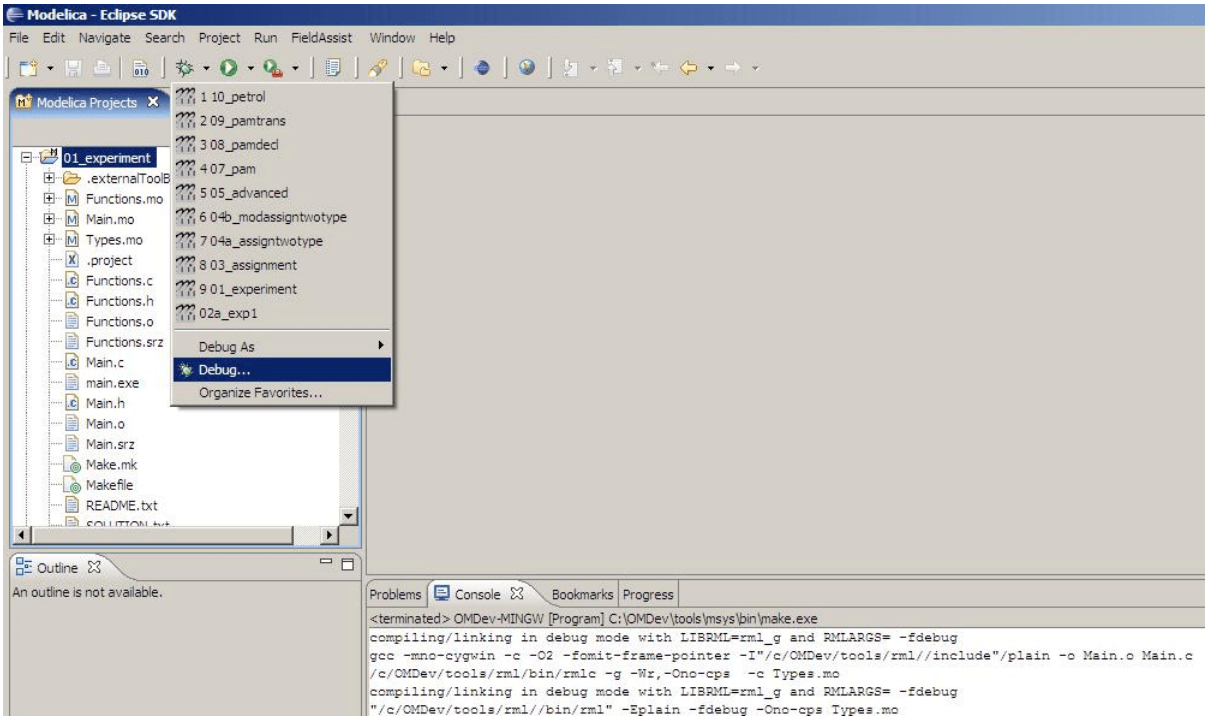


Figure 15.2: Accessing the debug configuration dialog.

To create the debug configuration, right click on the classification Modelica Development Tooling (MDT) GDB and select New as in figure below. Then give a name to the configuration, select the debugging executable to be executed and give it command line parameters. There are several tabs in which the user can select additional debug configuration settings like the environment in which the executable should be run.

Note that we require Gnu Debugger (GDB) for debugging session. We must specify the GDB location, also we must pass our script file as an argument to OMC.

Setting/Deleting Breakpoints

The Eclipse interface allows to add/remove breakpoints. At the moment only line number based breakpoints are supported. Other alternative to set the breakpoints is; function breakpoints.

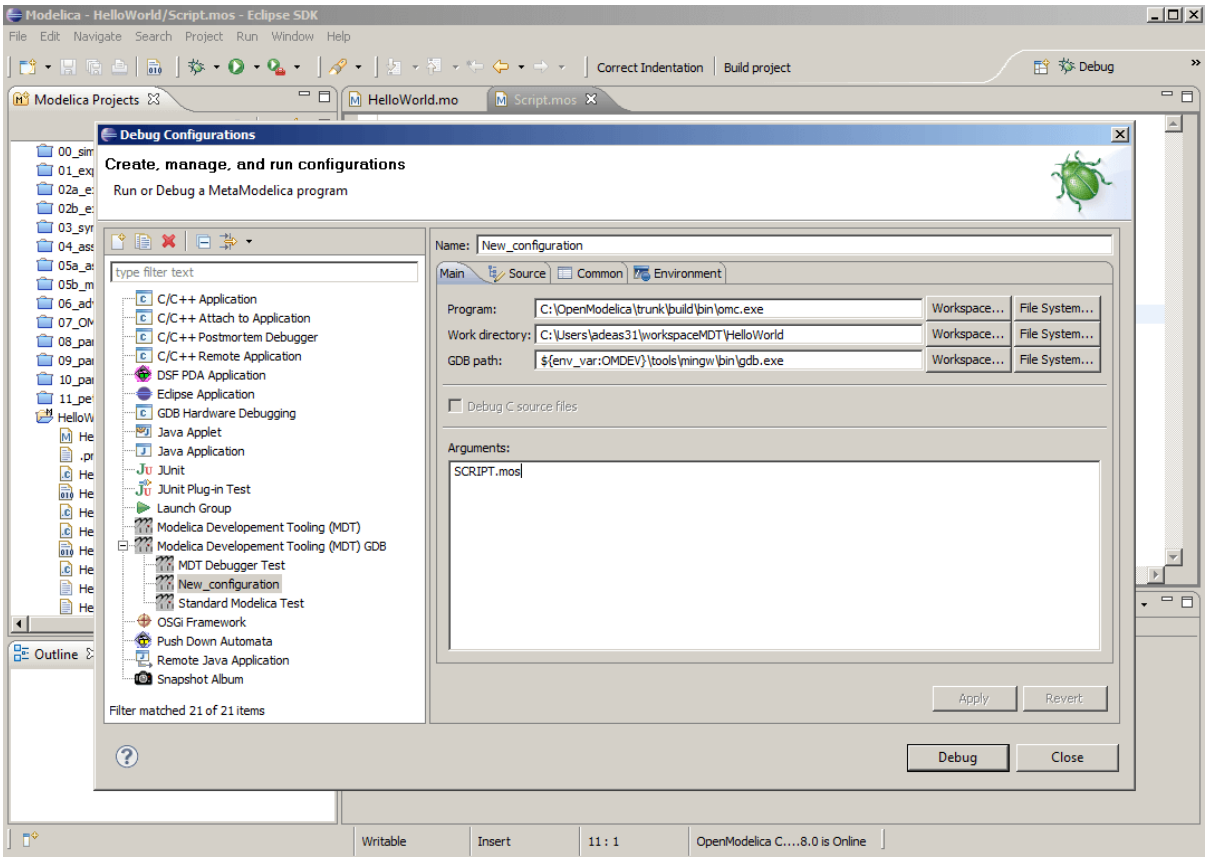


Figure 15.3: Creating the Debug Configuration.

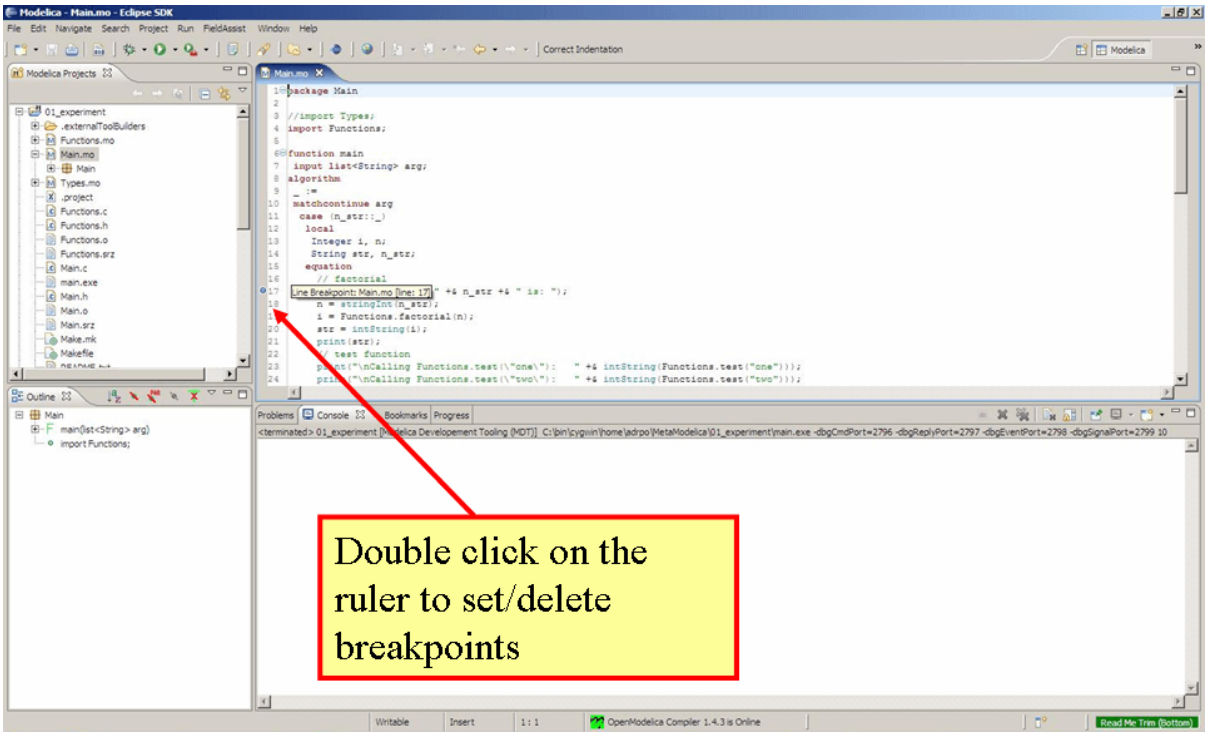


Figure 15.4: Setting/deleting breakpoints.

Starting the debugging session and enabling the debug perspective

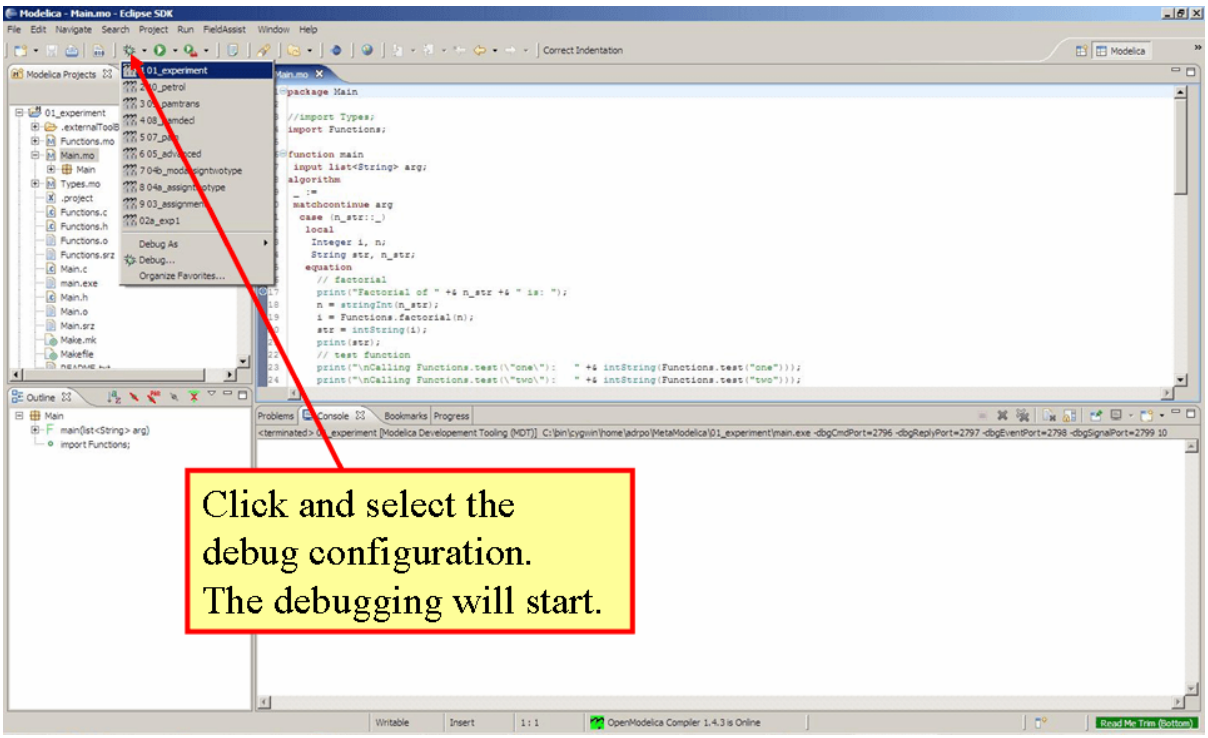


Figure 15.5: Starting the debugging session.

15.1.2 The Debugging Perspective

The debug view primarily consists of two main views:

- Stack Frames View
- Variables View

The stack frame view, shown in the figure below, shows a list of frames that indicates how the flow had moved from one function to another or from one file to another. This allows backtracing of the code. It is very much possible to select the previous frame in the stack and inspect the values of the variables in that frame. However, it is not possible to select any of the previous frame and start debugging from there. Each frame is shown as <function_name at file_name:line_number>.

The Variables view shows the list of variables at a certain point in the program, containing four colums:

- Name – the variable name.
- Declared Type – the Modelica type of the variable.
- Value – the variable value.
- Actual Type – the mapped C type.

By preserving the stack frames and variables it is possible to keep track of the variables values. If the value of any variable is changed while stepping then that variable will be highlighted yellow (the standard Eclipse way of showing the change).

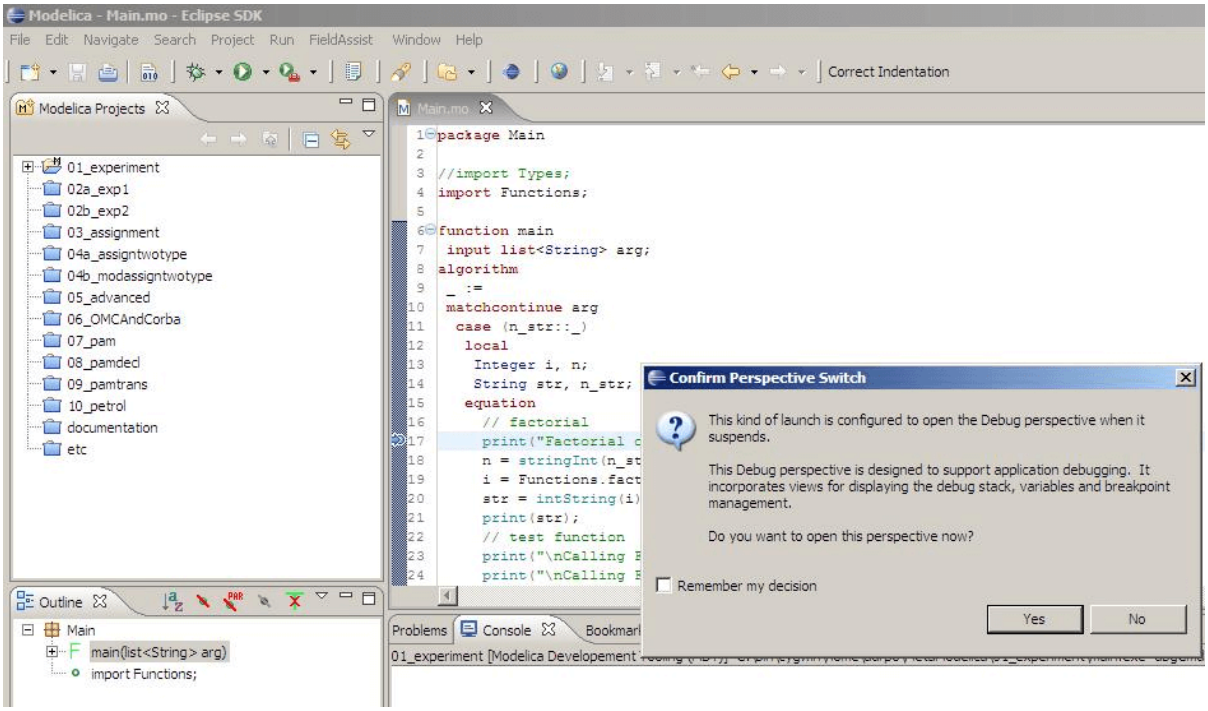


Figure 15.6: Eclipse will ask if the user wants to switch to the debugging perspective.

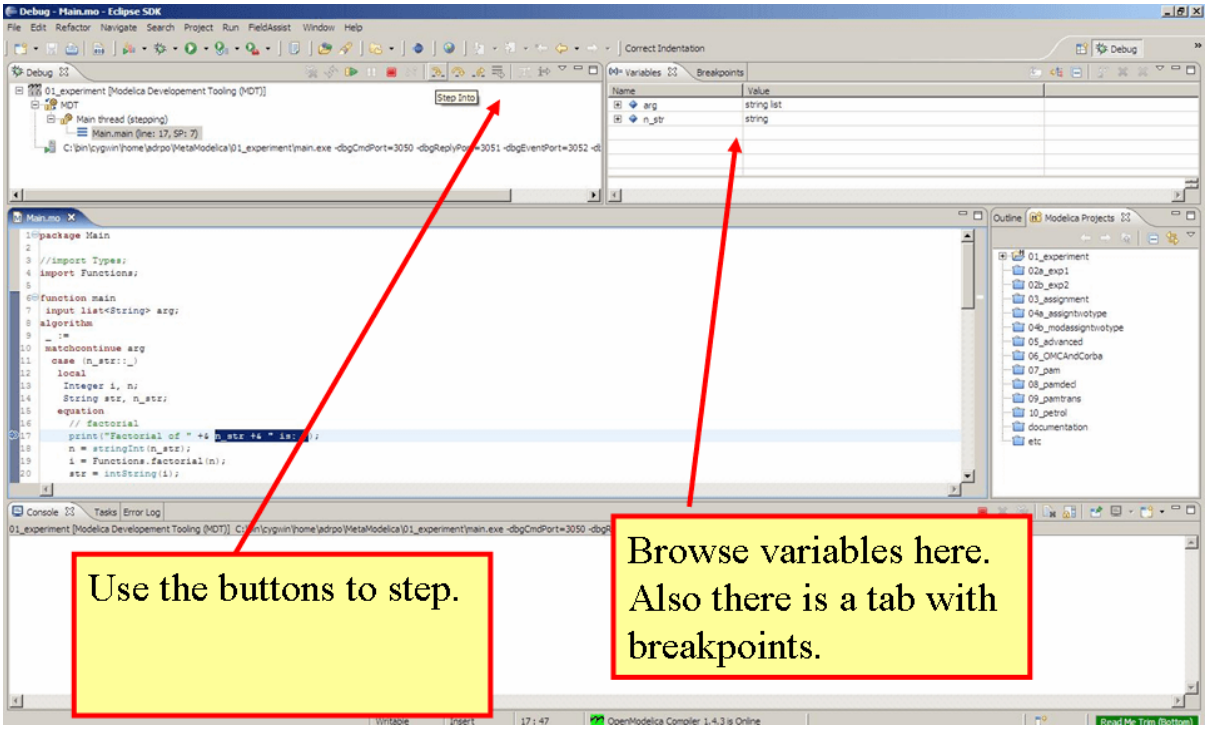


Figure 15.7: The debugging perspective.

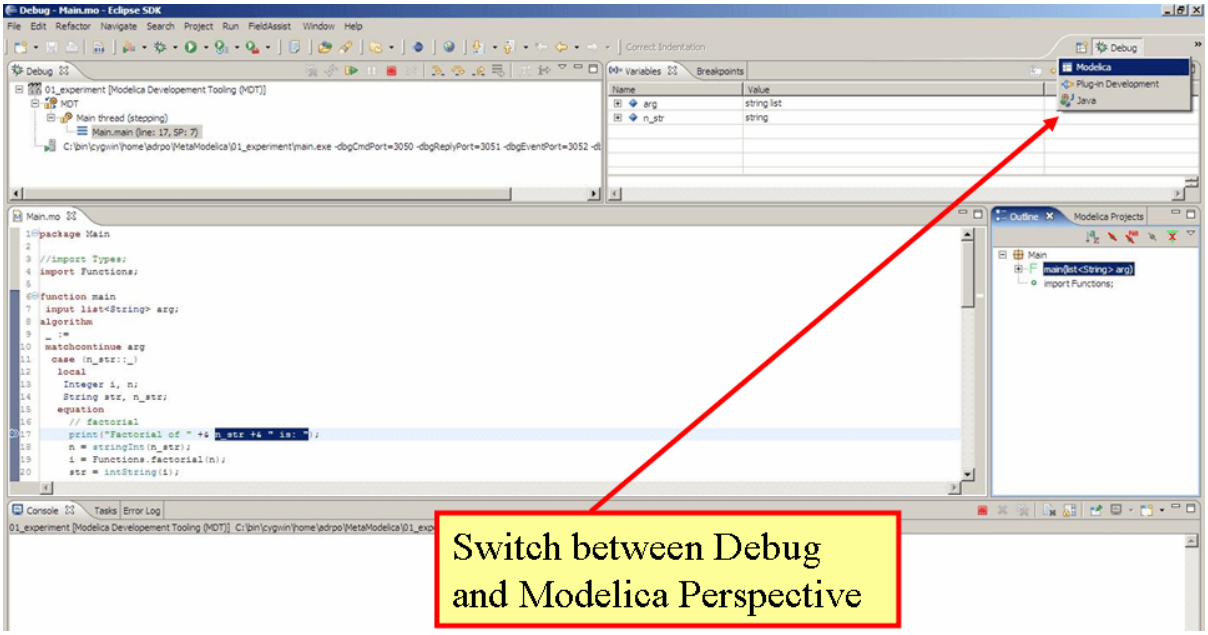


Figure 15.8: Switching between perspectives.

第16章　Modelica Performance Analyzer

A common problem when simulating models in an equation-based language like Modelica is that the model may contain non-linear equation systems. These are solved in each time-step by extrapolating an initial guess and running a non-linear system solver. If the simulation takes too long to simulate, it is useful to run the performance analysis tool. The tool has around 5~25% overhead, which is very low compared to instruction-level profilers (30x-100x overhead). Due to being based on a single simulation run, the report may contain spikes in the charts.

When running a simulation for performance analysis, execution times of user-defined functions as well as linear, non-linear and mixed equation systems are recorded.

To start a simulation in this mode, turn on profiling with the following command line flag >>> setCommandLineOptions("--profiling=all")

The generated report is in HTML format (with images in the SVG format), stored in a file modelname_prof.html, but the XML database and measured times that generated the report and graphs are also available if you want to customize the report for comparison with other tools.

Below we use the performance profiler on the simple model A:

```
model ProfilingTest
  function f
    input Real r;
    output Real o = sin(r);
  end f;
  String s = "abc";
  Real x = f(x) "This is x";
  Real y(start=1);
  Real z1 = cos(z2);
  Real z2 = sin(z1);
equation
  der(y) = time;
end ProfilingTest;
```

We simulate as usual, after setting the profiling flag:

```
>>> setCommandLineOptions("--profiling=blocks+html")
true
>>> simulate(ProfilingTest)
record SimulationResult
    resultFile = "<<DOCHOME>>/ProfilingTest_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 500,
 tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'ProfilingTest', options = '
↪', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
Warning: empty y range [1:1], adjusting to [0.99:1.01]
Warning: empty y range [1:1], adjusting to [0.99:1.01]
```

(次のページに続く)

(前のページからの続き)

```
Warning: empty y range [1:1], adjusting to [0.99:1.01]
Warning: empty y range [1:1], adjusting to [0.99:1.01]
Warning: empty y range [1:1], adjusting to [0.99:1.01]
Warning: empty y range [1:1], adjusting to [0.99:1.01]
stdout            | info    | Time measurements are stored in ProfilingTest_prof.
↪html (human-readable) and ProfilingTest_prof.xml (for XSL transforms or more␣
↪details)
",
    timeFrontend = 0.0150631,
    timeBackend = 0.0217501,
    timeSimCode = 0.0022029,
    timeTemplates = 0.0121642,
    timeCompile = 2.5497387,
    timeSimulation = 0.6822546,
    timeTotal = 3.284997
end SimulationResult;
"Warning: There are nonlinear iteration variables with default zero start␣
↪attribute found in NLSJac0. For more information set -d=initialization. In␣
↪OMEdit Tools->Options->Simulation->OMCFlags, in OMNotebook call␣
↪setCommandLineOptions("-d=initialization").
Warning: The initial conditions are not fully specified. For more information set -
↪d=initialization. In OMEdit Tools->Options->Simulation->OMCFlags, in OMNotebook␣
↪call setCommandLineOptions("-d=initialization").
"
```

16.1 Profiling information for ProfilingTest

16.1.1 Information

All times are measured using a real-time wall clock. This means context switching produces bad worst-case execution times (max times) for blocks. If you want better results, use a CPU-time clock or run the command using real-time priviliges (avoiding context switches).

Note that for blocks where the individual execution time is close to the accuracy of the real-time clock, the maximum measured time may deviate a lot from the average.

For more details, see ProfilingTest_prof.xml.

16.1.2 Settings

Name	Value
Integration method	dassl
Output format	mat
Output name	ProfilingTest_res.mat
Output size	24.0 kB
Profiling data	ProfilingTest_prof.data
Profiling size	0 B

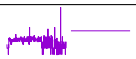
16.1.3 Summary

Task	Time	Fraction
Pre-Initialization	0.001839	3.49%
Initialization	0.001832	3.47%
Event-handling	0.000000	0.00%
Creating output file	0.005205	9.87%
Linearization		NaN%
Time steps	0.035559	67.42%
Overhead	0.002233	4.23%
Unknown	NaN	NaN%
Total simulation time	0.052745	100.00%

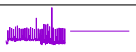
16.1.4 Global Steps

	Steps	Total Time	Fraction	Average Time	Max Time	Deviation
	499	0.035559	67.42%	0.0000712605210420842	0.000978966	12.74x

16.1.5 Measured Function Calls

	Name	Calls	Time	Fraction	Max Time	Deviation
	*ProfilingTest.f*	506	0.000601848	1.14%	0.000016066	12.51x

16.1.6 Measured Blocks

	Name	Calls	Time	Fraction	Max Time	Deviation
	` <#eq0>`__	7	0.001023264	1.94%	0.001031866	6.06x
	` <#eq12>`__	2	0.000020033	0.04%	0.000021466	1.14x
	` <#eq20>`__	504	0.005202715	9.86%	0.000090166	7.73x
	` <#eq22>`__	504	0.007198915	13.65%	0.000087166	5.10x

Equations

Name	Variables
eq0	
eq1	*y*
eq2	*s*
eq3	
eq4	*z2*
eq5	
eq6	` <#var0>`__
eq7	` <#var0>`__
eq8	` <#var0>`__
eq9	` <#var0>`__
eq10	*z1*
eq11	
eq12	*x*
eq13	*der(y)*
eq14	*z2*
eq15	
eq16	` <#var0>`__
eq17	` <#var0>`__
eq18	` <#var0>`__
eq19	` <#var0>`__
eq20	*z1*
eq21	
eq22	*x*
eq23	

Variables

Name	Comment
y	
der(y)	
x	This is x
z1	
z2	
s	

This report was generated by OpenModelica on 2020-12-14 00:19:27.

16.2 Genenerated JSON for the Example

Listing 16.1: ProfilingTest_prof.json


```
{
"name":"ProfilingTest",
"prefix":"ProfilingTest",
"date":"2020-12-14 00:19:27",
"method":"dassl",
"outputFormat":"mat",
"outputFilename":"ProfilingTest_res.mat",
"outputFilesize":24581,
"overheadTime":0.00236595,
"preinitTime":0.00183937,
"initTime":0.00183227,
"eventTime":0,
"outputTime":0.00520498,
"jacobianTime":0.000296393,
"totalTime":0.0527454,
"totalStepsTime":6.71663e-05,
"totalTimeProfileBlocks":0.0134449,
"numStep":499,
"maxTime":0.0009789663,
"functions":[
{"name":"ProfilingTest.f","ncall":506,"time":0.000601848,"maxTime":0.000016066}
],
"profileBlocks":[
{"id":0,"ncall":7,"time":0.001023264,"maxTime":0.001031866},
{"id":12,"ncall":2,"time":0.000020033,"maxTime":0.000021466},
{"id":20,"ncall":504,"time":0.005202715,"maxTime":0.000090166},
{"id":22,"ncall":504,"time":0.007198915,"maxTime":0.000087166}
]
}
```


16.3 Using the Profiler from OMEdit

When running a simulation from OMEdit, it is possible to enable profiling information, which can be combined with the *transformations browser*.

When profiling the DoublePendulum example from MSL, the following output in Figure 16.2 is a typical result. This information clearly shows which system takes longest to simulate (a linear system, where most of the time overhead probably comes from initializing LAPACK over and over).

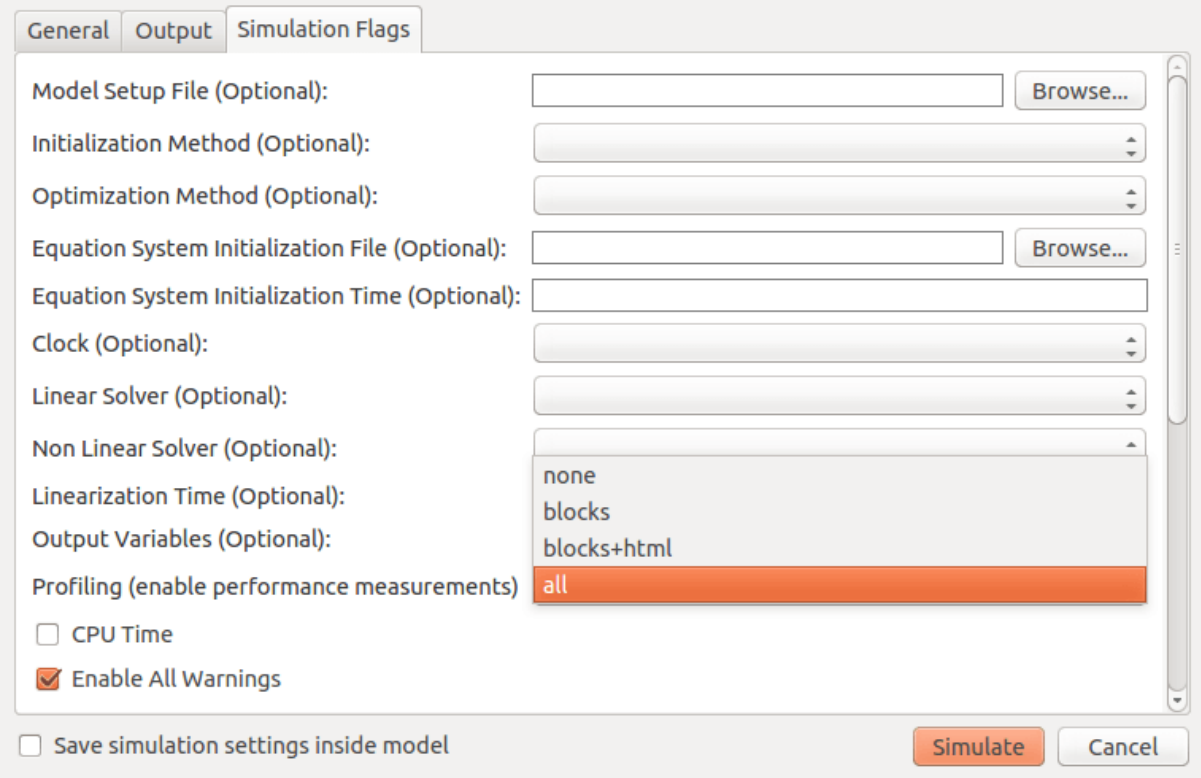


Figure 16.1: Setting up the profiler from OMEdit.

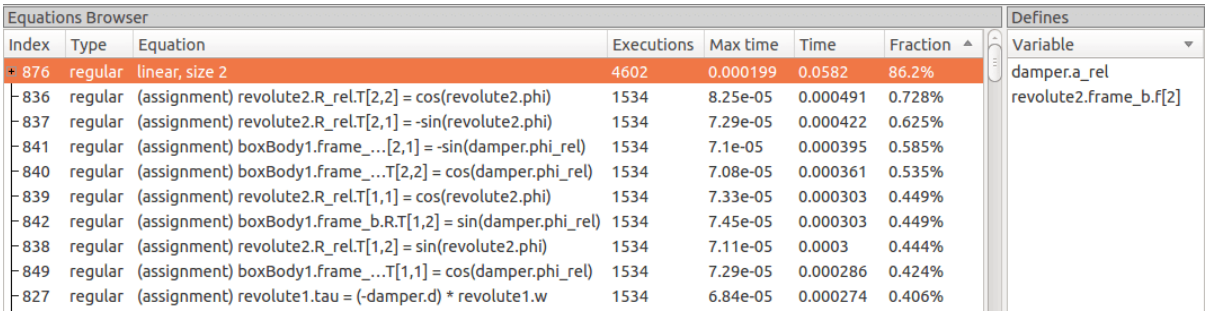


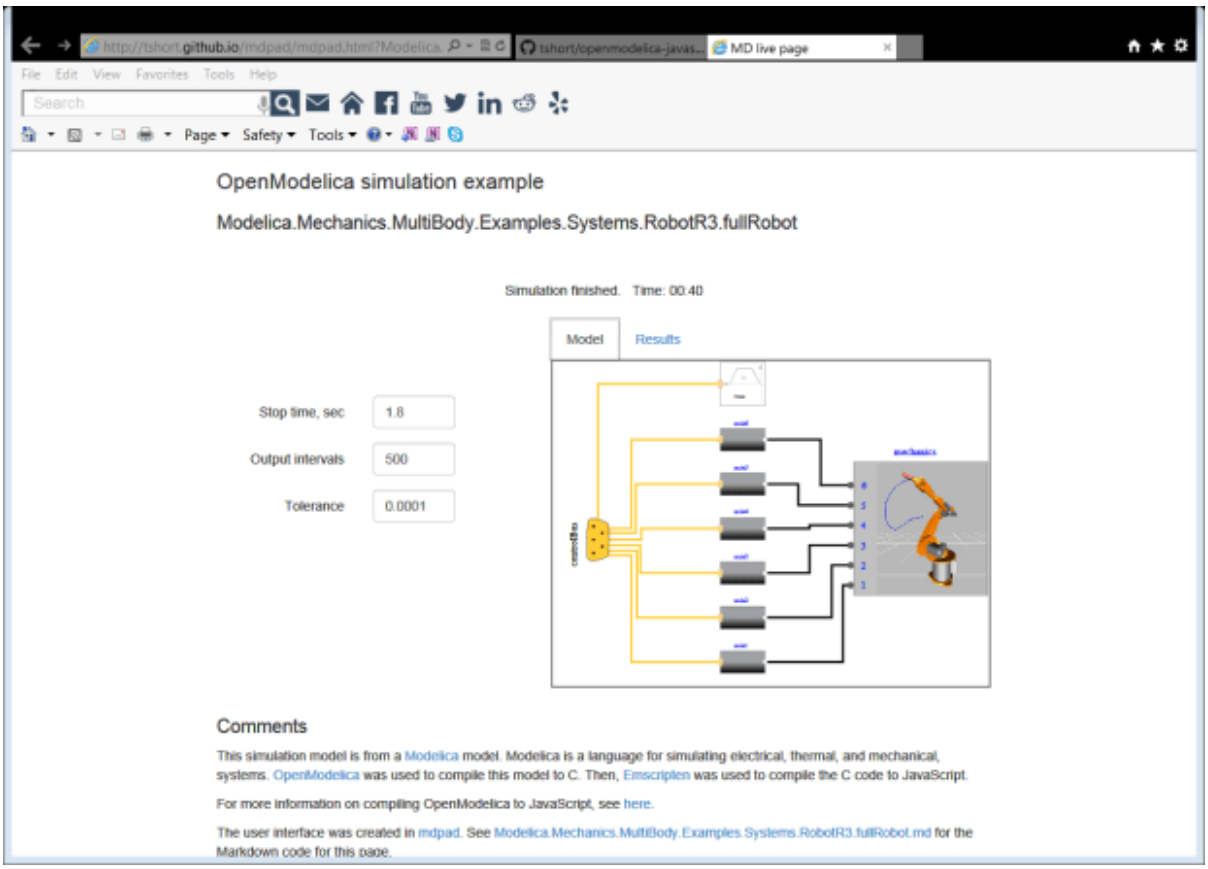
Figure 16.2: Profiling results of the Modelica standard library DoublePendulum example, sorted by execution time.

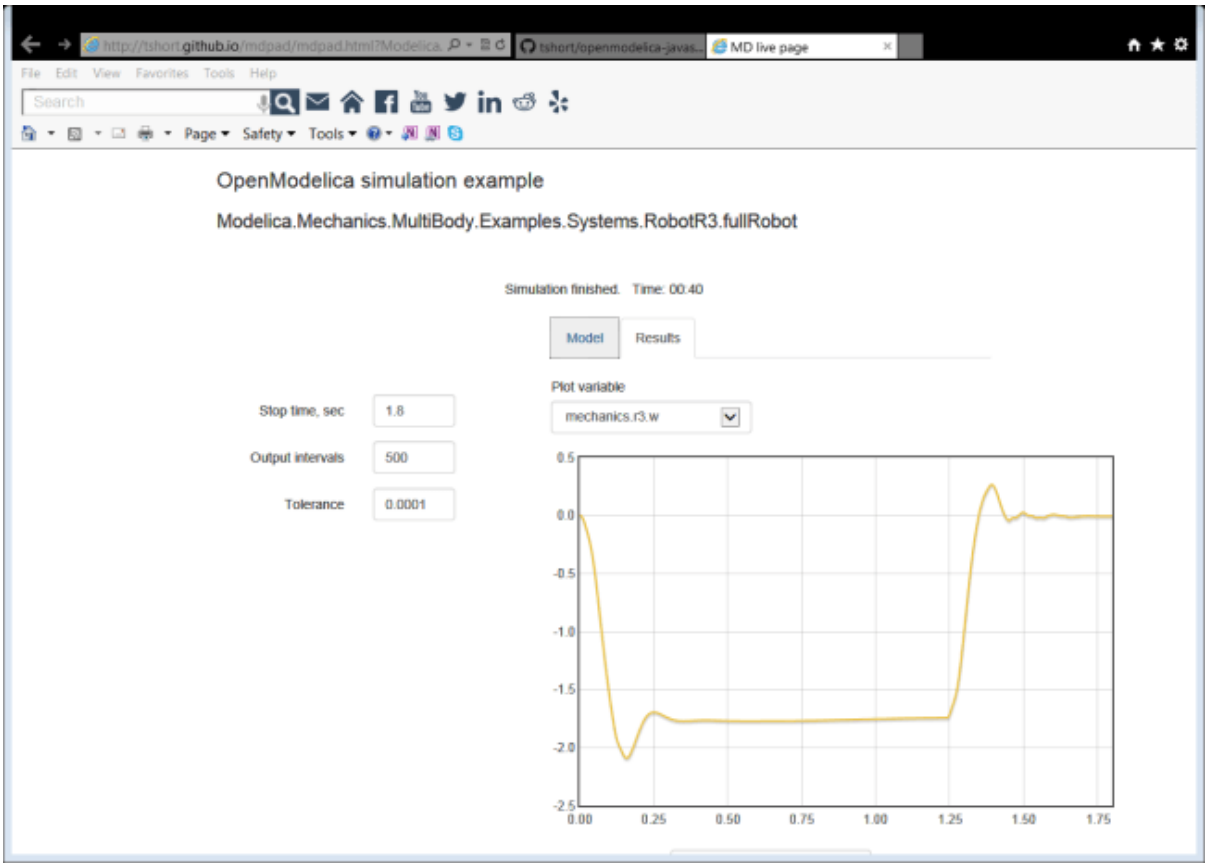
第17章　Simulation in Web Browser

OpenModelica can simulate in a web browser on a client computer by model code being compiled to efficient Javacript code.

For more information, see https://github.com/tshort/openmodelica-javascript

Below used on the MSL MultiBody RobotR3.fullRobot example model.




MD live page
OpenModelica simulation example
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot
Simulation finished. Time: 00:40
Model
Results
Stop time, sec
1.8
Output intervals
500
Tolerance
0.0001
Plot variable
mechanics.r3.w
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-2.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75

第18章　Interoperability – C and Python

Below is information and examples about the OpenModelica external C interfaces, as well as examples of Python interoperability.

18.1 Calling External C functions

The following is a small example (ExternalLibraries.mo) to show the use of external C functions:

```
model ExternalLibraries

  function ExternalFunc1
    input Real x;
    output Real y;
  external y=ExternalFunc1_ext(x) annotation(Library="ExternalFunc1.o",␣
↪LibraryDirectory="modelica://ExternalLibraries", Include="#include \
↪"ExternalFunc1.h\"");
  end ExternalFunc1;

  function ExternalFunc2
    input Real x;
    output Real y;
  external "C" annotation(Library="ExternalFunc2", LibraryDirectory="modelica://
↪ExternalLibraries");
  end ExternalFunc2;

  Real x(start=1.0, fixed=true), y(start=2.0, fixed=true);
equation
  der(x)=-ExternalFunc1(x);
  der(y)=-ExternalFunc2(y);
end ExternalLibraries;
```

These C (.c) files and header files (.h) are needed (note that the headers are not needed since OpenModelica will generate the correct definition if it is not present; using the headers it is possible to write C-code directly in the Modelica source code or declare non-standard calling conventions):

Listing 18.1: ExternalFunc1.c

```
double ExternalFunc1_ext(double x)
{
  double res;
  res = x+2.0*x*x;
  return res;
}
```

Listing 18.2: ExternalFunc1.h

```
double ExternalFunc1_ext(double);
```

Listing 18.3: ExternalFunc2.c

```
double ExternalFunc2(double x)
{
  double res;
  res = (x-1.0)*(x+2.0);
  return res;
}
```

The following script file ExternalLibraries.mos will perform everything that is needed, provided you have gcc installed in your path:

```
>>> system(getCompiler() + " -c -o ExternalFunc1.o ExternalFunc1.c")
0
>>> system(getCompiler() + " -c -o ExternalFunc2.o ExternalFunc2.c")
0
>>> system("ar rcs libExternalFunc2.a ExternalFunc2.o")
0
>>> simulate(ExternalLibraries)
record SimulationResult
    resultFile = "<<DOCHOME>>/ExternalLibraries_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 500,␣
↪tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'ExternalLibraries',␣
↪options = '', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags␣
↪= ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.006689400000000001,
    timeBackend = 0.0060286,
    timeSimCode = 0.0022494,
    timeTemplates = 0.0118945,
    timeCompile = 2.6167011,
    timeSimulation = 0.1168799,
    timeTotal = 2.7627249
end SimulationResult;
```

And plot the results:

18.2 Calling external Python Code from a Modelica model

The following calls external Python code through a very simplistic external function (no data is retrieved from the Python code). By making it a dynamically linked library, you might get the code to work without changing the linker settings.

```
function pyRunString
  input String s;
external "C" annotation(Include="
#include <Python.h>
```

(次のページに続く)

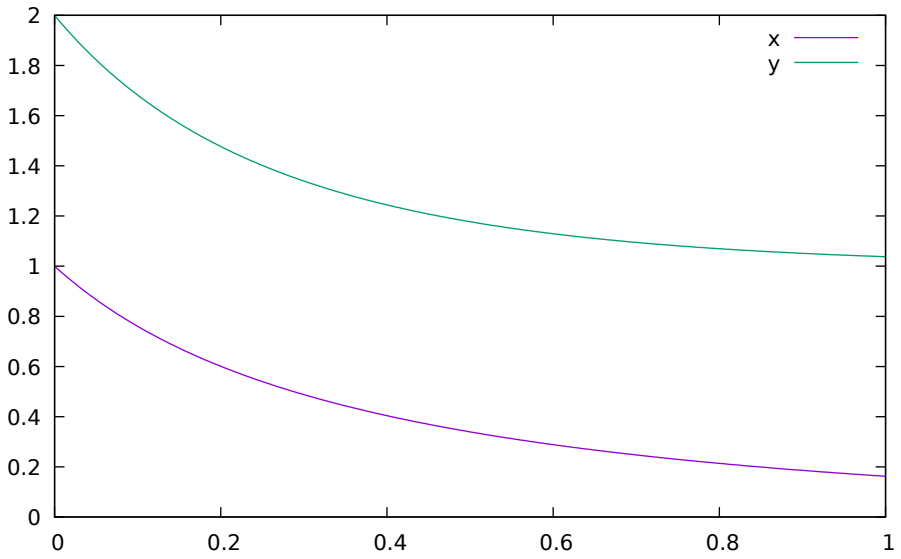


Figure 18.1: Plot generated by OpenModelica+gnuplot

(前のページからの続き)

```
void pyRunString(const char *str)
{
  Py_SetProgramName(\"pyRunString\");  /* optional but recommended */
  Py_Initialize();
  PyRun_SimpleString(str);
  Py_Finalize();
}
");
end pyRunString;

model CallExternalPython
algorithm
  pyRunString("
print 'Python says: simulation time',"+String(time)+"
");
end CallExternalPython;
```

```
>>> system("python-config --cflags > pycflags")
0
>>> system("python-config --ldflags > pyldflags")
0
>>> pycflags := stringReplace(readFile("pycflags"),"\n","");
>>> pyldflags := stringReplace(readFile("pyldflags"),"\n","");
>>> setCFlags(getCFlags()+pycflags)
true
>>> setLinkerFlags(getLinkerFlags()+pyldflags)
true
>>> simulate(CallExternalPython, stopTime=2)
record SimulationResult
    resultFile = "<<DOCHOME>>/CallExternalPython_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 2.0, numberOfIntervals = 500,␣
↪tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'CallExternalPython',␣
↪options = '', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags␣
↪= ''",
    messages = "Python says: simulation time 0
```

(次のページに続く)

(前のページからの続き)

```
Python says: simulation time 0
LOG_SUCCESS       | info    | The initialization finished successfully without␣
↪homotopy method.
Python says: simulation time 2
LOG_SUCCESS       | info    | The simulation finished successfully.
",
    timeFrontend = 0.0066116,
    timeBackend = 0.0111104,
    timeSimCode = 0.0009169,
    timeTemplates = 0.0086117,
    timeCompile = 2.9144268,
    timeSimulation = 0.3587216999999999,
    timeTotal = 3.3022596
end SimulationResult;
```

18.3 Calling OpenModelica from Python Code

This section describes a simple-minded approach to calling Python code from OpenModelica. For a description of Python scripting with OpenModelica, see *OMPython – OpenModelica Python Interface*.

The interaction with Python can be perfomed in four different ways whereas one is illustrated below. Assume that we have the following Modelica code:

Listing 18.4: CalledbyPython.mo

```
model CalledbyPython
  Real x(start=1.0), y(start=2.0);
  parameter Real b = 2.0;
equation
  der(x) = -b*y;
  der(y) = x;
end CalledbyPython;
```

In the following Python (.py) files the above Modelica model is simulated via the OpenModelica scripting interface:

Listing 18.5: PythonCaller.py

```
#!/usr/bin/python
import sys,os
global newb = 0.5
execfile('CreateMosFile.py')
os.popen(r"omc CalledbyPython.mos").read()
execfile('RetrResult.py')
```

Listing 18.6: CreateMosFile.py

```
#!/usr/bin/python
mos_file = open('CalledbyPython.mos','w', 1)
mos_file.write('loadFile("CalledbyPython.mo");\n')
mos_file.write('setComponentModifierValue(CalledbyPython,b,$Code(="+str(newb)+"));\
↪n')
mos_file.write('simulate(CalledbyPython,stopTime=10);\n')
mos_file.close()
```

Listing 18.7: RetrResult.py

```
#!/usr/bin/python
def zeros(n): #
  vec = [0.0]
  for i in range(int(n)-1): vec = vec + [0.0]
  return vec
res_file = open("CalledbyPython_res.plt",'r',1)
line = res_file.readline()
size = int(res_file.readline().split('=')[1])
time = zeros(size)
y = zeros(size)
while line != ['DataSet: time\\n']:
  line = res_file.readline().split(',')[0:1]
for j in range(int(size)):
  time[j]=float(res\_file.readline().split(',')[0])
while line != ['DataSet: y\\n']:
  line=res_file.readline().split(',')[0:1]
for j in range(int(size)):
  y[j]=float(res\_file.readline().split(',')[1])
res_file.close()
```

A second option of simulating the above Modelica model is to use the command buildModel instead of the simulate command and setting the parameter value in the initial parameter file, CalledbyPython_init.txt instead of using the command setComponentModifierValue. Then the file CalledbyPython.exe is just executed.

The third option is to use the Corba interface for invoking the compiler and then just use the scripting interface to send commands to the compiler via this interface.

The fourth variant is to use external function calls to directly communicate with the executing simulation process.

第19章 OpenModelica Python Interface and PySimulator

This chapter describes the OpenModelica Python integration facilities.

- OMPython – the OpenModelica Python scripting interface, see *OMPython – OpenModelica Python Interface*.
- EnhancedOMPython - Enhanced OMPython scripting interface, see *Enhanced OMPython Features*.
- PySimulator – a Python package that provides simulation and post processing/analysis tools integrated with OpenModelica, see *PySimulator*.

19.1 OMPython – OpenModelica Python Interface

OMPython – OpenModelica Python API is a free, open source, highly portable Python based interactive session handler for Modelica scripting. It provides the modeler with components for creating a complete Modelica modeling, compilation and simulation environment based on the latest OpenModelica library standard available. OMPython is architectured to combine both the solving strategy and model building. So domain experts (people writing the models) and computational engineers (people writing the solver code) can work on one unified tool that is industrially viable for optimization of Modelica models, while offering a flexible platform for algorithm development and research. OMPython is not a standalone package, it depends upon the OpenModelica installation.

OMPython is implemented in Python and depends either on the OmniORB and OmniORBpy - high performance CORBA ORBs for Python or ZeroMQ - high performance asynchronous messaging library and it supports the Modelica Standard Library version 3.2 that is included in starting with OpenModelica 1.9.2.

To install OMPython follow the instructions at https://github.com/OpenModelica/OMPython

19.1.1 Features of OMPython

OMPython provides user friendly features like:

- Interactive session handling, parsing, interpretation of commands and Modelica expressions for evaluation, simulation, plotting, etc.
- Interface to the latest OpenModelica API calls.
- Optimized parser results that give control over every element of the output.
- Helper functions to allow manipulation on Nested dictionaries.

- Easy access to the library and testing of OpenModelica commands.

19.1.2 Test Commands

OMPython provides two classes for communicating with OpenModelica i.e., OMCSession and OMCSessionZMQ. Both classes have the same interface, the only difference is that OMCSession uses omniORB and OMCSessionZMQ uses ZeroMQ. All the examples listed down uses OMCSessionZMQ but if you want to test OMCSession simply replace OMCSessionZMQ with OMCSession. We recommend to use OMCSessionZMQ.

To test the command outputs, simply create an OMCSessionZMQ object by importing from the OMPython library within Python interepreter. The module allows you to interactively send commands to the OMC server and display their output.

To get started, create an OMCSessionZMQ object:

```
>>> from OMPython import OMCSessionZMQ
>>> omc = OMCSessionZMQ()
```

```
>>> omc.sendExpression("getVersion()")
OMCompiler v1.14.1
>>> omc.sendExpression("cd()")
<<DOCHOME>>
>>> omc.sendExpression("loadModel(Modelica)")
True
>>> omc.sendExpression("loadFile(getInstallationDirectoryPath() + \"/share/doc/omc/
↪testmodels/BouncingBall.mo\")")
True
>>> omc.sendExpression("instantiateModel(BouncingBall)")
class BouncingBall
  parameter Real e = 0.7 "coefficient of restitution";
  parameter Real g = 9.81 "gravity acceleration";
  Real h(start = 1.0, fixed = true) "height of ball";
  Real v(fixed = true) "velocity of ball";
  Boolean flying(start = true, fixed = true) "true, if ball is flying";
  Boolean impact;
  Real v_new(fixed = true);
  Integer foo;
equation
  impact = h <= 0.0;
  foo = if impact then 1 else 2;
  der(v) = if flying then -g else 0.0;
  der(h) = v;
  when {h <= 0.0 and v <= 0.0, impact} then
    v_new = if edge(impact) then (-e) * pre(v) else 0.0;
    flying = v_new > 0.0;
    reinit(v, v_new);
  end when;
end BouncingBall;
```

We get the name and other properties of a class:

```
>>> omc.sendExpression("getClassNames()")
('BouncingBall', 'ModelicaServices', 'Complex', 'Modelica')
>>> omc.sendExpression("isPartial(BouncingBall)")
False
>>> omc.sendExpression("isPackage(BouncingBall)")
```

(次のページに続く)

(前のページからの続き)

```
False
>>> omc.sendExpression("isModel(BouncingBall)")
True
>>> omc.sendExpression("checkModel(BouncingBall)")
Check of BouncingBall completed successfully.
Class BouncingBall has 6 equation(s) and 6 variable(s).
1 of these are trivial equation(s).
>>> omc.sendExpression("getClassRestriction(BouncingBall)")
model
>>> omc.sendExpression("getClassInformation(BouncingBall)")
('model', '', False, False, False, '/home/OpenModelica/build/share/doc/omc/
↪testmodels/BouncingBall.mo', False, 1, 1, 23, 17, (), False, False, '', '',␣
↪False, '')
>>> omc.sendExpression("getConnectionCount(BouncingBall)")
0
>>> omc.sendExpression("getInheritanceCount(BouncingBall)")
0
>>> omc.sendExpression("getComponentModifierValue(BouncingBall,e)")
0.7
>>> omc.sendExpression("checkSettings()")
{'OPENMODELICAHOME': '<<OPENMODELICAHOME>>', 'OPENMODELICALIBRARY': '<
↪<OPENMODELICAHOME>>/lib/omlibrary', 'OMC_PATH': '<<OPENMODELICAHOME>>/bin/omc',
↪'SYSTEM_PATH': '/home/.local/bin:/opt/ThirdParty-6/platforms/linux64Gcc/
↪gperftools-svn/bin:/opt/paraviewopenfoam56/bin:/home/OpenFOAM/uedashige-6/
↪platforms/linux64GccDPInt32Opt/bin:/opt/site/6/platforms/linux64GccDPInt32Opt/
↪bin:/opt/openfoam6/platforms/linux64GccDPInt32Opt/bin:/opt/openfoam6/bin:/opt/
↪openfoam6/wmake:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
↪usr/games:/usr/local/games:/mnt/c/Program:Files/WindowsApps/
↪CanonicalGroupLimited.UbuntuonWindows_2004.2020.812.0_x64__79rhkp1fndgsc:/mnt/c/
↪Program:Files/Microsoft:MPI/Bin/:/mnt/c/Program:Files:(x86)/Common:Files/Apple/
↪Apple:Application:Support:/mnt/c/WINDOWS/system32:/mnt/c/WINDOWS:/mnt/c/WINDOWS/
↪System32/Wbem:/mnt/c/WINDOWS/System32/WindowsPowerShell/v1.0/:/mnt/c/WINDOWS/
↪System32/OpenSSH/:/mnt/c/Program:Files:(x86)/Fujitsu/FJAgent/Core/bin:/mnt/c/
↪Program:Files:(x86)/Elmer:8.3-432ee0dd/bin:/mnt/c/Program:Files/gs/gs9.23/bin:/
↪mnt/c/Program:Files/Git/cmd:/mnt/c/Program:Files/TortoiseSVN/bin:/mnt/c/
↪Program:Files:(x86)/GitExtensions/:/mnt/c/Users/dabes/AppData/Local/Microsoft/
↪WindowsApps:/mnt/c/Program:Files:(x86)/Elmer:8.2-Release/bin:/mnt/c/Users/dabes/
↪AppData/Local/Programs/Microsoft:VS:Code/bin:/snap/bin', 'OMDEV_PATH': '', 'OMC_
↪FOUND': True, 'MODELICAUSERCFLAGS': '', 'WORKING_DIRECTORY': '<<DOCHOME>>',
↪'CREATE_FILE_WORKS': True, 'REMOVE_FILE_WORKS': True, 'OS': 'linux', 'SYSTEM_INFO
↪': 'Linux LAPTOP-8Q5NG2D4 4.4.0-18362-Microsoft #1049-Microsoft Thu Aug 14␣
↪12:01:00 PST 2020 x86_64 x86_64 x86_64 GNU/Linux\n', 'RTLIBS': ' -Wl,--no-as-
↪needed -Wl,--disable-new-dtags -lOpenModelicaRuntimeC  -llapack -lblas   -lm -
↪lomcgc -lpthread -rdynamic', 'C_COMPILER': 'clang', 'C_COMPILER_VERSION': 'clang␣
↪version 6.0.0-1ubuntu2 (tags/RELEASE_600/final)\nTarget: x86_64-pc-linux-gnu\
↪nThread model: posix\nInstalledDir: /usr/bin\n', 'C_COMPILER_RESPONDING': True,
↪'HAVE_CORBA': False, 'CONFIGURE_CMDLINE': "Configured 2020-04-22 22:17:47 using␣
↪arguments:  '--disable-option-checking' '--prefix=/home/work/OM/OMBuild/20200101_
↪2/OpenModelica/build' 'CC=clang' 'CXX=clang++' '-without-omc' '-with-cppruntime'
↪'--with-ombuilddir=/home/work/OM/OMBuild/20200101_2/OpenModelica/build' '--cache-
↪file=/dev/null' '--srcdir=.'"}
```

The common combination of a simulation followed by getting a value and doing a plot:

```
>>> omc.sendExpression("simulate(BouncingBall, stopTime=3.0)")
{'resultFile': '<<DOCHOME>>/BouncingBall_res.mat', 'simulationOptions': "startTime␣
↪= 0.0, stopTime = 3.0, numberOfIntervals = 500, tolerance = 1e-06, method =
↪'dassl', fileNamePrefix = 'BouncingBall', options = '', outputFormat = 'mat',
 variableFilter = '.*', cflags = '', simflags = ''", 'messages': 'LOG_SUCCESS
 | info    | The initialization finished successfully without homotopy method.\
↪nLOG_SUCCESS       | info    | The simulation finished successfully.\n',
↪'timeFrontend': 0.5558981, 'timeBackend': 0.0085836, 'timeSimCode': 0.002548,
 'timeTemplates': 0.0124056, 'timeCompile': 2.5286644, 'timeSimulation': 0.1387271,
 'timeTotal': 3.2499372}
```

(次のページに続く)

(前のページからの続き)

```
>>> omc.sendExpression("val(h , 2.0)")
0.04239430772884106
```

Import As Library

To use the module from within another python program, simply import OMCSessionZMQ from within the using program.

For example:

```
# test.py
from OMPython import OMCSessionZMQ
omc = OMCSessionZMQ()
cmds = [
  'loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/
↪BouncingBall.mo")',
  "simulate(BouncingBall)",
  "plot(h)"
  ]
for cmd in cmds:
  answer = omc.sendExpression(cmd)
  print("\n{}:\n{}".format(cmd, answer))
```

19.1.3 Implementation

Client Implementation

The OpenModelica Python API Interface – OMPython, attempts to mimic the OMShell's style of operations.

OMPython is designed to,

- Initialize the CORBA/ZeroMQ communication.
- Send commands to the OMC server via the CORBA/ZeroMQ interface.
- Receive the string results.
- Use the Parser module to format the results.
- Return or display the results.

19.2 Enhanced OMPython Features

Some more improvements are added to OMPython functionality for querying more information about the models and simulate them. A list of new user friendly API functionality allows user to extract information about models using python objects. A list of API functionality is described below.

To get started, create a ModelicaSystem object:

```
>>> from OMPython import OMCSessionZMQ
>>> omc = OMCSessionZMQ()
>>> model_path=omc.sendExpression("getInstallationDirectoryPath()") + "/share/doc/
↪omc/testmodels/"
>>> from OMPython import ModelicaSystem
>>> mod=ModelicaSystem(model_path + "BouncingBall.mo","BouncingBall")
```

The object constructor requires a minimum of 2 input arguments which are strings, and may need a third string input argument.

- The first input argument must be a string with the file name of the Modelica code, with Modelica file extension ".mo". If the Modelica file is not in the current directory of Python, then the file path must also be included.
- The second input argument must be a string with the name of the Modelica model including the namespace if the model is wrapped within a Modelica package.
- The third input argument is used to specify the list of dependent libraries or dependent Modelica files e.g.,

```
>>> mod=ModelicaSystem(model_path + "BouncingBall.mo","BouncingBall",["Modelica"])
```

- By default ModelicaSystem uses OMCSessionZMQ but if you want to use OMCSession then pass the argument *useCorba=True* to the constructor.

19.2.1 BuildModel

The buildModel API can be used after ModelicaSystem(), in case the model needs to be updated or additional simulationflags needs to be set using sendExpression()

```
>>> mod.buildModel()
```

19.2.2 Standard get methods

- getQuantities()
- getContinuous()
- getInputs()
- getOutputs()
- getParameters()

- getSimulationOptions()
- getSolutions()

Three calling possibilities are accepted using getXXX() where "XXX" can be any of the above functions (eg:) getParameters().

- getXXX() without input argument, returns a dictionary with names as keys and values as values.
- getXXX(S), where S is a string of names.
- getXXX(["S1","S2"]) where S1 and S1 are list of string elements

19.2.3 Usage of getMethods

```
>>> mod.getQuantities() // method-1, list of all variables from xml file
[{'aliasvariable': None, 'Name': 'height', 'Variability': 'continuous', 'Value':
↪'1.0', 'alias': 'noAlias', 'Changeable': 'true', 'Description': None}, {
↪'aliasvariable': None, 'Name': 'c', 'Variability': 'parameter', 'Value': '0.9',
↪'alias': 'noAlias', 'Changeable': 'true', 'Description': None}]
```

```
>>> mod.getQuantities("height") // method-2, to query information about single␣
↪quantity
[{'aliasvariable': None, 'Name': 'height', 'Variability': 'continuous', 'Value':
↪'1.0', 'alias': 'noAlias', 'Changeable': 'true', 'Description': None}]
```

```
>>> mod.getQuantities(["c","radius"]) // method-3, to query information about list␣
↪of quantity
[{'aliasvariable': None, 'Name': 'c', 'Variability': 'parameter', 'Value': '0.9',
 'alias': 'noAlias', 'Changeable': 'true', 'Description': None}, {'aliasvariable':␣
↪None, 'Name': 'radius', 'Variability': 'parameter', 'Value': '0.1', 'alias':
↪'noAlias', 'Changeable': 'true', 'Description': None}]
```

```
>>> mod.getContinuous() // method-1, list of continuous variable
{'velocity': -1.825929609047952, 'der(velocity)': -9.8100000000000005, 'der(height)
↪': -1.825929609047952, 'height': 0.65907039052943617}
```

```
>>> mod.getContinuous(["velocity","height"]) // method-2, get specific variable␣
↪value information
(-1.825929609047952, 0.65907039052943617)
```

```
>>> mod.getInputs()
{}
```

```
>>>  mod.getOutputs()
{}
```

```
>>> mod.getParameters()  // method-1
{'c': 0.9, 'radius': 0.1}
```

```
>>> mod.getParameters(["c","radius"]) // method-2
[0.9, 0.1]
```

```
>>> mod.getSimulationOptions()  // method-1
{'stepSize': 0.002, 'stopTime': 1.0, 'tolerance': 1e-06, 'startTime': 0.0, 'solver
↪': 'dassl'}
```

```
>>> mod.getSimulationOptions(["stepSize","tolerance"]) // method-2
[0.002, 1e-06]
```

The getSolution method can be used in two different ways.

1) using default result filename

2) use the result filenames provided by user

This provides a way to compare simulation results and perform regression testing

```
>>> mod.getSolutions() // method-1 returns list of simulation variables for which␣
↪results are available
['time', 'height', 'velocity', 'der(height)', 'der(velocity)', 'c', 'radius']
```

```
>>> mod.getSolutions(["time","height"])  // return list of numpy arrays
```

```
>>> mod.getSolutions(resultfile="c:/tmpbouncingBall.mat") // method-2 returns list␣
↪of simulation variables for which results are available , the resulfile location␣
↪is provided by user
```

```
>>> mod.getSolutions(["time","height"],resultfile="c:/tmpbouncingBall.mat") //␣
↪return list of array
```

19.2.4 Standard set methods

- setInputs()
- setParameters()
- setSimulationOptions()

Two setting possibilities are accepted using setXXXs(),where "XXX" can be any of above functions.

- setXXX("Name=value") string of keyword assignments
- setXXX(["Name1=value1","Name2=value2","Name3=value3"]) list of string of keyword assignments

19.2.5 Usage of setMethods

```
>>> mod.setInputs(["cAi=1","Ti=2"]) // method-2
```

```
>>> mod.setParameters("radius=14") // method-1 setting parameter value
```

```
>>> mod.setParameters(["radius=14","c=0.5"]) // method-2 setting parameter value␣
↪using second option
```

```
>>> mod.setSimulationOptions(["stopTime=2.0","tolerance=1e-08"]) // method-2
```

19.2.6 Simulation

An example of how to get parameter names and change the value of parameters using set methods and finally simulate the "BouncingBall.mo" model is given below.

```
>>>  mod.getParameters()
{'c': 0.9, 'radius': 0.1}
```

```
>>>  mod.setParameters(["radius=14","c=0.5"]) //setting parameter value
```

To check whether new values are updated to model , we can again query the getParameters().

```
>>> mod.getParameters()
{'c': 0.5, 'radius': 14}
```

And then finally we can simulate the model using, The simulate() API can be used in two methods

1) without any arguments

2) resultfile names provided by user (only filename is allowed and not the location)

```
>>> mod.simulate() // method-1 default result file name will be used
>>> mod.simulate(resultfile="tmpbouncingBall.mat")  // method-2 resultfile name␣
↪provided by users
```

19.2.7 Linearization

The following methods are proposed for linearization.

- linearize()
- getLinearizationOptions()
- setLinearizationOptions()
- getLinearInputs()
- getLinearOutputs()
- getLinearStates()

19.2.8 Usage of Linearization methods

```
>>> mod.getLinearizationOptions()  // method-1
{'simflags': ' ', 'stepSize': 0.002, 'stopTime': 1.0, 'startTime': 0.0,
↪'numberOfIntervals': 500.0, 'tolerance': 1e-08}
```

```
>>> mod.getLinearizationOptions("startTime","stopTime") // method-2
[0.0, 1.0]
```

```
>>> mod.setLinearizationOptions(["stopTime=2.0","tolerance=1e-06"])
```

```
>>> mod.linearize()  //returns a tuple of 2D numpy arrays (matrices) A, B, C and D.
```

```
>>> mod.getLinearInputs()  //returns a list of strings of names of inputs used␣
↪when forming matrices.
```

```
>>> mod.getLinearOutputs() //returns a list of strings of names of outputs used␣
↪when forming matrices
```

```
>>> mod.getLinearStates() // returns a list of strings of names of states used␣
↪when forming matrices.
```

19.3 PySimulator

PySimulator provides a graphical user interface for performing analyses and simulating different model types (currently Functional Mockup Units and Modelica Models are supported), plotting result variables and applying simulation result analysis tools like Fast Fourier Transform.

Read more about the PySimulator at https://github.com/PySimulator/PySimulator.

第20章　OMMatlab – OpenModelica Matlab Interface

OMMatlab – the OpenModelica Matlab API is a free, open source, highly portable Matlab-based interactive session handler for Modelica scripting. It provides the modeler with components for creating a complete Modelica modeling, compilation and simulation environment based on the latest OpenModelica library standard available. OMMatlab is architectured to combine both the solving strategy and model building. So domain experts (people writing the models) and computational engineers (people writing the solver code) can work on one unified tool that is industrially viable for optimization of Modelica models, while offering a flexible platform for algorithm development and research. OMMatlab is not a standalone package, it depends upon the OpenModelica installation.

OMMatlab is implemented in Matlab and depends on ZeroMQ - high performance asynchronous messaging library and it supports the Modelica Standard Library version 3.2 that is included in starting with OpenModelica 1.9.2.

To install OMMatlab follow the instructions at https://github.com/OpenModelica/OMMatlab

20.1 Features of OMMatlab

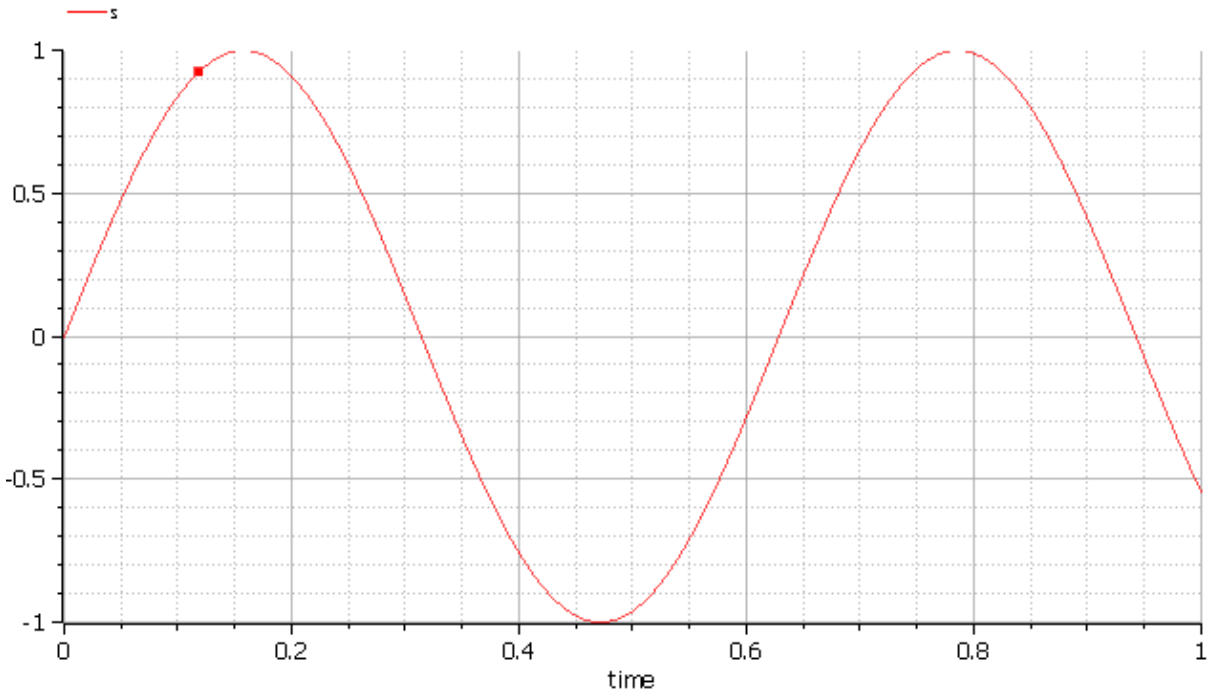
The OMMatlab package contains the following features:

- Import the OMMatlab package in Matlab
- Connect with the OpenModelica compiler through zmq sockets
- Able to interact with the OpenModelica compiler through the *available API*
- All the API calls are communicated with the help of the sendExpression method implemented in a Matlab pacakge
- The results are returned as strings

20.2 Test Commands

To get started, create a OMMatlab session object:

```
>>> import OMMatlab.*
>>> omc= OMMatlab()
>>> omc.sendExpression("getVersion()")
'v1.13.0-dev-531-gde26b558a (64-bit)'
>>> omc.sendExpression("loadModel(Modelica)")
'true'
>>> omc.sendExpression("model a Real s; equation s=sin(10*time); end a;")
'{a}'
>>> omc.sendExpression("simulate(a)")
>>> omc.sendExpression("plot(s)")
'true'
```



20.2.1 Advanced OMMatlab Features

OMMatlab package has advanced functionality for querying more information about the models and simulate them. A list of new user friendly API functionality allows user to extract information about models using matlab objects. A list of API functionality is described below.

To get started, create a ModelicaSystem object:

```
>>> import OMMatlab.*
>>> omc= OMMatlab()
>>> omc.ModelicaSystem("BouncingBall.mo","BouncingBall")
```

The object constructor requires a minimum of 2 input arguments which are strings, and third input argument which is optional .

- The first input argument must be a string with the file name of the Modelica code, with Modelica file extension ".mo". If the Modelica file is not in the current directory, then the file path must also be included.
- The second input argument must be a string with the name of the Modelica model including the namespace if the model is wrapped within a Modelica package.
- The third input argument (optional) is used to specify the list of dependent libraries or dependent Modelica files The argument can be passed as a string or array of strings e.g.,

```
>>> omc.ModelicaSystem("BouncingBall.mo","BouncingBall",["Modelica",
↪"SystemDynamics", "dcmotor.mo"])
```

20.3 WorkDirectory

For each Matlab session a temporary work directory is created and the results are published in that working directory, Inorder to get the workdirectory the users can use the following API

```
>>> omc.getWorkDirectory()
'C:/Users/arupa54/AppData/Local/Temp/tp7dd648e5_5de6_4f66_b3d6_90bce1fe1d58'
```

20.4 BuildModel

The buildModel API can be used after ModelicaSystem(), in case the model needs to be updated or additional simulationflags needs to be set using sendExpression()

```
>>> buildModel(mod)
```

20.5 Standard get methods

- getQuantities()
- showQuantities()
- getContinuous()
- getInputs()
- getOutputs()
- getParameters()
- getSimulationOptions()
- getSolutions()

Three calling possibilities are accepted using getXXX() where "XXX" can be any of the above functions (eg:) getParameters().

- getXXX() without input argument, returns a dictionary with names as keys and values as values.

- getXXX(S), where S is a string of names.
- getXXX(["S1","S2"]) where S1 and S1 are array of string elements

20.6 Usage of getMethods

```
>>> omc.getQuantities() // method-1, list of all variables from xml file
+----------+------------+--------------------------+--------------+------------+----
↪-------+---------------+--------+
| name     | changeable | description              | variability  | causality  |␣
↪alias     | aliasVariable | value |
+----------+------------+--------------------------+--------------+------------+----
↪-------+---------------+--------+
| 'h'      | 'true'     | 'height of ball'         | 'continuous' | 'internal' |
↪'noAlias' | ''            | '1.0' |
+----------+------------+--------------------------+--------------+------------+----
↪-------+---------------+--------+
| 'v'      | 'true'     | 'velocity of ball'       | 'continuous' | 'internal' |
↪'noAlias' | ''            | ''    |
+----------+------------+--------------------------+--------------+------------+----
↪-------+---------------+--------+
| 'der(h)' | 'false'    | 'der(height of ball)'    | 'continuous' | 'internal' |
↪'noAlias' | ''            | ''    |
+----------+------------+--------------------------+--------------+------------+----
↪-------+---------------+--------+
| 'der(v)' | 'false'    | 'der(velocity of ball)' | 'continuous' | 'internal' |
↪'noAlias' | ''            | ''    |
+----------+------------+--------------------------+--------------+------------+----
↪-------+---------------+--------+
```

```
>>> omc.getQuantities("h") // method-2, to query information about single quantity
+----------+------------+--------------------------+--------------+------------+----
↪-------+---------------+--------+
| name     | changeable | description              | variability  | causality  |␣
↪alias     | aliasVariable | value |
+----------+------------+--------------------------+--------------+------------+----
↪-------+---------------+--------+
| 'h'      | 'true'     | 'height of ball'         | 'continuous' | 'internal' |
↪'noAlias' | ''            | '1.0' |
+----------+------------+--------------------------+--------------+------------+----
↪-------+---------------+--------+
```

```
>>> omc.getQuantities(["h","v"]) // method-3, to query information about list of␣
↪quantity
+----------+------------+--------------------------+--------------+------------+----
↪-------+---------------+--------+
| name     | changeable | description              | variability  | causality  |␣
↪alias     | aliasVariable | value |
+----------+------------+--------------------------+--------------+------------+----
↪-------+---------------+--------+
| 'h'      | 'true'     | 'height of ball'         | 'continuous' | 'internal' |
↪'noAlias' | ''            | '1.0' |
+----------+------------+--------------------------+--------------+------------+----
↪-------+---------------+--------+
| 'v'      | 'true'     | 'velocity of ball'       | 'continuous' | 'internal' |
↪'noAlias' | ''            | ''    |
+----------+------------+--------------------------+--------------+------------+----
↪-------+---------------+--------+
```

```
>>> omc.getContinuous() // method-1, returns struct of continuous variable
struct with fields:
  h     : '1.0'
  v     : ''
  der_h_: ''
  der_v_: ''
```

```
>>> omc.getContinuous(["h","v"])    // method-2, returns string array
"1.0"    ""
```

```
>>> omc.getInputs()
struct with no fields
```

```
>>> omc.getOutputs()
struct with no fields
```

```
>>> omc.getParameters()  // method-1
struct with fields:
        e: '0.7'
        g: '9.810000000000001'
```

```
>>> omc.getParameters(["c","radius"]) // method-2
"0.7"  "9.810000000000001"
```

```
>>> omc.getSimulationOptions()  // method-1
struct with fields:
        startTime: '0'
         stopTime: '1'
         stepSize: '0.002'
        tolerance: '1e-006'
           solver: 'dassl'
```

```
>>> omc.getSimulationOptions(["stepSize","tolerance"]) // method-2
"0.002", "1e-006"
```

The getSolution method can be used in two different ways.

1) using default result filename

2) use the result filenames provided by user

This provides a way to compare simulation results and perform regression testing

```
>>> omc.getSolutions() // method-1 returns string arrays of simulation variables␣
↪for which results are available, the default result filename is taken
"time", "height", ""velocity", "der(height)", "der(velocity)", "c", "radius"
```

```
>>> omc.getSolutions(["time","h"])  // return list of cell arrays
1 × 2 cell array
{1 × 506 double}    {1 × 506 double}
```

```
>>> omc.getSolutions([],"c:/tmpbouncingBall.mat") // method-2 returns string␣
↪arrays of simulation variables for which results are available , the resulfile␣
↪location is provided by user
"time", "height", "velocity", "der(height)", "der(velocity)", "c", "radius"
```

```
>>> omc.getSolutions(["time","h"],"c:/tmpbouncingBall.mat") // return list of cell␣
↪arrays
1 × 2 cell array
{1 × 506 double}    {1 × 506 double}
```

20.7 Standard set methods

- setInputs()
- setParameters()
- setSimulationOptions()

Two setting possibilities are accepted using setXXXs(),where "XXX" can be any of above functions.

- setXXX("Name=value") string of keyword assignments
- setXXX(["Name1=value1","Name2=value2","Name3=value3"]) array of string of keyword assignments

20.8 Usage of setMethods

```
>>> omc.setInputs("cAi=1") // method-1
```

```
>>> omc.setInputs(["cAi=1","Ti=2"]) // method-2
```

```
>>> omc.setParameters("e=14") // method-1
```

```
>>> omc.setParameters(["e=14","g=10.8"]) // method-2 setting parameter value using␣
↪array of string
```

```
>>> omc.setSimulationOptions(["stopTime=2.0","tolerance=1e-08"])
```

20.9 Advanced Simulation

An example of how to do advanced simulation to set parameter values using set methods and finally simulate the "BouncingBall.mo" model is given below .

```
>>> omc.getParameters()
struct with fields:
        e: '0.7'
        g: '9.810000000000001'
```

```
>>> omc.setParameters(["e=0.9","g=9.83"])
```

To check whether new values are updated to model , we can again query the getParameters().

```
>>> omc.getParameters()
struct with fields:
    e: "0.9"
    g: "9.83"
```

Similary we can also use setInputs() to set a value for the inputs during various time interval can also be done using the following.

```
>>> omc.setInputs("cAi=1")
```

And then finally we can simulate the model using, The simulate() API can be used in two methods

1) without any arguments

2) resultfile names provided by user (only filename is allowed and not the location)

```
>>> omc.simulate() // method-1 default result file name will be used
>>> omc.simulate("tmpbouncingBall.mat") // method-2 resultfile name provided by␣
↪users
```

20.10 Linearization

The following methods are available for linearization of a modelica model

- linearize()
- getLinearizationOptions()
- setLinearizationOptions()
- getLinearInputs()
- getLinearOutputs()
- getLinearStates()

20.11 Usage of Linearization methods

```
>>> omc.getLinearizationOptions()  // method-1
```

```
>>> omc.getLinearizationOptions(["startTime","stopTime"]) // method-2
"0.0", "1.0"
```

```
>>> omc.setLinearizationOptions(["stopTime=2.0","tolerance=1e-08"])
```

```
>>> omc.linearize()  //returns a list 2D arrays (matrices) A, B, C and D.
```

```
>>> omc.getLinearInputs()  //returns a list of strings of names of inputs used␣
↪when forming matrices.
```

```
>>> omc.getLinearOutputs() //returns a list of strings of names of outputs used␣
↪when forming matrices.
```

```
>>> omc.getLinearStates() // returns a list of strings of names of states used␣
↪when forming matrices.
```

第21章　OMJulia – OpenModelica Julia Scripting

OMJulia – the OpenModelica Julia API is a free, open source, highly portable Julia based interactive session handler for Julia scripting of OpenModelica API functionality. It provides the modeler with components for creating a complete Julia-Modelica modeling, compilation and simulation environment based on the latest OpenModelica implementation and Modelica library standard available. OMJulia is architectured to combine both the solving strategy and model building. Thus, domain experts (people writing the models) and computational engineers (people writing the solver code) can work on one unified tool that is industrially viable for optimization of Modelica models, while offering a flexible platform for algorithm development and research. OMJulia is not a standalone package, it depends upon the OpenModelica installation.

OMJulia is implemented in Julia and depends on ZeroMQ - high performance asynchronous messaging library and it supports the Modelica Standard Library version 3.2 that is included in starting with OpenModelica 1.9.2.

To install OMJulia follow the instructions at https://github.com/OpenModelica/OMJulia.jl

21.1 Features of OMJulia

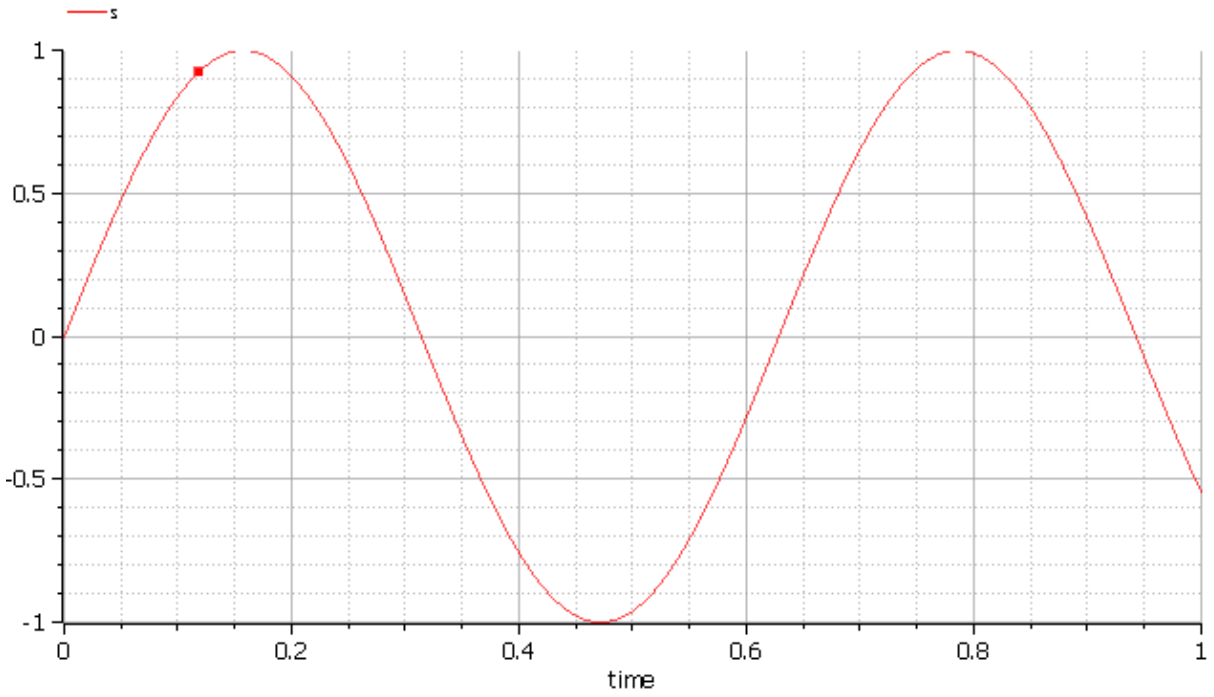
The OMJulia package contains the following features:

- Interactive session handling, parsing, interpretation of commands and Modelica expressions for evaluation, simulation, plotting, etc.
- Connect with the OpenModelica compiler through zmq sockets
- Able to interact with the OpenModelica compiler through the *available API*
- Easy access to the Modelica Standard library.
- All the API calls are communicated with the help of the sendExpression method implemented in a Julia module
- The results are returned as strings

21.2 Test Commands

To get started, create an OMJulia session object:

```
>>> using OMJulia
>>> omc= OMJulia.OMCSession()
>>> sendExpression(omc,"loadModel(Modelica)")
true
>>> sendExpression(omc,"model a Real s; equation s=sin(10*time); end a;")
1-element Array{Symbol,1}:
 :a
>>> sendExpression(omc,"simulate(a)")
>>> sendExpression(omc,"plot(s)")
true
```



21.2.1 Advanced OMJulia Features

OMJulia package has advanced functionality for querying more information about the models and simulate them. A list of new user friendly API functionality allows user to extract information about models using julia objects. A list of API functionality is described below.

To get started, create a ModelicaSystem object:

```
>>> using OMJulia
>>> mod = OMJulia.OMCSession()
>>> ModelicaSystem(mod,"BouncingBall.mo","BouncingBall")
```

The object constructor requires a minimum of 2 input arguments which are strings, and third input argument which is optional .

- The first input argument must be a string with the file name of the Modelica code, with Modelica file extension ".mo". If the Modelica file is not in the current directory, then the file path must also be included.
- The second input argument must be a string with the name of the Modelica model including the namespace if the model is wrapped within a Modelica package.
- The third input argument (optional) is used to specify the list of dependent libraries or dependent Modelica files The argument can be passed as a string or array of strings e.g.,

```
>>> ModelicaSystem(mod,"BouncingBall.mo","BouncingBall",["Modelica",
↪"SystemDynamics", "dcmotor.mo"])
```

21.3 WorkDirectory

For each OMJulia session a temporary work directory is created and the results are published in that working directory, Inorder to get the workdirectory the users can use the following API

```
>>> getWorkDirectory(mod)
"C:/Users/arupa54/AppData/Local/Temp/jl_5pbewl"
```

21.4 BuildModel

The buildModel API can be used after ModelicaSystem(), in case the model needs to be updated or additional simulationflags needs to be set using sendExpression()

```
>>> buildModel(mod)
```

21.5 Standard get methods

- getQuantities()
- showQuantities()
- getContinuous()
- getInputs()
- getOutputs()
- getParameters()
- getSimulationOptions()
- getSolutions()

Three calling possibilities are accepted using getXXX() where "XXX" can be any of the above functions (eg:) getParameters().

- getXXX() without input argument, returns a dictionary with names as keys and values as values.

- getXXX(S), where S is a string of names.
- getXXX(["S1","S2"]) where S1 and S1 are array of string elements

21.6 Usage of getMethods

```
>>> getQuantities(mod) // method-1, list of all variables from xml file
[{"aliasvariable": None, "Name": "height", "Variability": "continuous", "Value":
↪"1.0", "alias": "noAlias", "Changeable": "true", "Description": None}, {
↪"aliasvariable": None, "Name": "c", "Variability": "parameter", "Value": "0.9",
↪"alias": "noAlias", "Changeable": "true", "Description": None}]
```

```
>>> getQuantities(mod,"height") // method-2, to query information about single␣
↪quantity
[{"aliasvariable": None, "Name": "height", "Variability": "continuous", "Value":
↪"1.0", "alias": "noAlias", "Changeable": "true", "Description": None}]
```

```
>>> getQuantities(mod,["c","radius"]) // method-3, to query information about list␣
↪of quantity
[{"aliasvariable": None, "Name": "c", "Variability": "parameter", "Value": "0.9",
 "alias": "noAlias", "Changeable": "true", "Description": None}, {"aliasvariable":␣
↪None, "Name": "radius", "Variability": "parameter", "Value": "0.1", "alias":
↪"noAlias", "Changeable": "true", "Description": None}]
```

```
>>> getContinuous(mod) // method-1, list of continuous variable
{"velocity": "-1.825929609047952", "der(velocity)": "-9.8100000000000005",
↪"der(height)": "-1.825929609047952", "height": "0.65907039052943617"}
```

```
>>> getContinuous(mod,["velocity","height"]) // method-2, get specific variable␣
↪value information
["-1.825929609047952", "0.65907039052943617"]
```

```
>>> getInputs(mod)
{}
```

```
>>> getOutputs(mod)
{}
```

```
>>> getParameters(mod)  // method-1
{"c": "0.9", "radius": "0.1"}
```

```
>>> getParameters(mod,["c","radius"]) // method-2
["0.9", "0.1"]
```

```
>>> getSimulationOptions(mod)  // method-1
{"stepSize": "0.002", "stopTime": "1.0", "tolerance": "1e-06", "startTime": "0.0",
↪"solver": "dassl"}
```

```
>>> getSimulationOptions(mod,["stepSize","tolerance"]) // method-2
["0.002", "1e-06"]
```

The getSolution method can be used in two different ways.

1) using default result filename

2) use the result filenames provided by user

This provides a way to compare simulation results and perform regression testing

```
>>> getSolutions(mod) // method-1 returns list of simulation variables for which␣
↪results are available
["time", "height", ""velocity", "der(height)", "der(velocity)", "c", "radius"]
```

```
>>> getSolutions(mod,["time","height"])  // return list of array
```

```
>>> getSolutions(mod,resultfile="c:/tmpbouncingBall.mat") // method-2 returns list␣
↪of simulation variables for which results are available , the resulfile location␣
↪is provided by user
["time", "height", ""velocity", "der(height)", "der(velocity)", "c", "radius"]
```

```
>>> getSolutions(mod,["time","h"],resultfile="c:/tmpbouncingBall.mat") // return␣
↪list of array
```

```
>>> showQuantities(mod) // same as getQuantities() but returns the results in the␣
↪form table
```

21.7 Standard set methods

- setInputs()
- setParameters()
- setSimulationOptions()

Two setting possibilities are accepted using setXXXs(),where "XXX" can be any of above functions.

- setXXX("Name=value") string of keyword assignments
- setXXX(["Name1=value1","Name2=value2","Name3=value3"]) array of string of keyword assignments

21.8 Usage of setMethods

```
>>> setInputs(mod,"cAi=1") // method-1
```

```
>>> setInputs(mod,["cAi=1","Ti=2"]) // method-2
```

```
>>> setParameters(mod,"radius=14") // method-1
```

```
>>> setParameters(mod,["radius=14","c=0.5"]) // method-2 setting parameter value␣
↪using array of string
```

```
>>> setSimulationOptions(mod,["stopTime=2.0","tolerance=1e-08"])
```

21.9 Advanced Simulation

An example of how to do advanced simulation to set parameter values using set methods and finally simulate the "BouncingBall.mo" model is given below .

```
>>> getParameters(mod)
{"c": "0.9", "radius": "0.1"}
```

```
>>> setParameters(mod,["radius=14","c=0.5"])
```

To check whether new values are updated to model , we can again query the getParameters().

```
>>> getParameters(mod)
{"c": "0.5", "radius": "14"}
```

Similary we can also use setInputs() to set a value for the inputs during various time interval can also be done using the following.

```
>>> setInputs(mod,"cAi=1")
```

And then finally we can simulate the model using, The simulate() API can be used in two methods

1) without any arguments

2) resultfile names provided by user (only filename is allowed and not the location)

```
>>> simulate(mod) // method-1 default result file name will be used
>>> simulate(mod,resultfile="tmpbouncingBall.mat") // method-2 resultfile name␣
↪provided by users
```

21.10 Linearization

The following methods are available for linearization of a modelica model

- linearize()
- getLinearizationOptions()
- setLinearizationOptions()
- getLinearInputs()
- getLinearOutputs()
- getLinearStates()

21.11 Usage of Linearization methods

```
>>> getLinearizationOptions(mod)  // method-1
{"stepSize": "0.002", "stopTime": "1.0", "startTime": "0.0", "numberOfIntervals":
↪"500.0", "tolerance": "1e-08"}
```

```
>>> getLinearizationOptions(mod,["startTime","stopTime"]) // method-2
["0.0", "1.0"]
```

```
>>> setLinearizationOptions(mod,["stopTime=2.0","tolerance=1e-06"])
```

```
>>> linearize(mod)  //returns a list 2D arrays (matrices) A, B, C and D.
```

```
>>> getLinearInputs(mod)  //returns a list of strings of names of inputs used when␣
↪forming matrices.
```

```
>>> getLinearOutputs(mod) //returns a list of strings of names of outputs used␣
↪when forming matrices.
```

```
>>> getLinearStates(mod) // returns a list of strings of names of states used when␣
↪forming matrices.
```

21.12 Sensitivity Analysis

A Method for computing numeric sensitivity of modelica model is available .

- (res1,res2) = sensitivity(arg1,arg2,arg3)

The constructor requires a minimum of 3 input arguments .

- arg1: Array of strings of Modelica Parameter names
- arg2: Array of strings of Modelica Variable names
- arg3: Array of float Excitations of parameters; defaults to scalar 1e-2

The results contains the following .

- res1: Vector of Sensitivity names.
- res2: Array of sensitivies: vector of elements per parameter, each element containing time series per variable.

21.13 Usage

```
>>> (Sn, Sa) = sensitivity(mod,["UA","EdR"],["T","cA"],[1e-2,1e-4])
```

With the above list of API calls implemented, the users can have more control over the result types, returned as Julia data structures.

第22章　Jupyter-OpenModelica

An OpenModelica Kernel for Jupyter Notebook. All commands are interpreted by OMPython which communicates with OpenModelica Compiler and the results are presented to user.

The project is available at https://github.com/OpenModelica/jupyter-openmodelica.

Follow the Readme file to install and start running modelica models directly in Jupyter Notebook

第23章　Scripting API

The following are short summaries of OpenModelica scripting commands. These commands are useful for loading and saving classes, reading and storing data, plotting of results, and various other tasks.

The arguments passed to a scripting function should follow syntactic and typing rules for Modelica and for the scripting function in question. In the following tables we briefly indicate the types or character of the formal parameters to the functions by the following notation:

- String typed argument, e.g. "hello", "myfile.mo".
- **TypeName – class, package or function name, e.g. MyClass,** Modelica.Math.
- VariableName – variable name, e.g. `v1`, `v2`, `vars1[2].x`, etc.
- Integer or Real typed argument, e.g. 35, 3.14, xintvariable.
- options – optional parameters with named formal parameter passing.

23.1 OpenModelica Scripting Commands

The following are brief descriptions of the scripting commands available in the OpenModelica environment. All commands are shown in alphabetical order:

23.1.1 interactiveDumpAbsynToJL

```
function interactiveDumpAbsynToJL
  output String res;
end interactiveDumpAbsynToJL;
```

23.1.2 relocateFunctions

```
function relocateFunctions
  input String fileName;
  input String names[:, 2];
  output Boolean success;
end relocateFunctions;
```

23.1.3 toJulia

```
function toJulia
  output String res;
end toJulia;
```

23.1.4 GC_expand_hp

```
function GC_expand_hp
  input Integer size;
  output Boolean success;
end GC_expand_hp;
```

23.1.5 GC_gcollect_and_unmap

23.1.6 GC_get_prof_stats

```
function GC_get_prof_stats
  output GC_PROFSTATS gcStats;
end GC_get_prof_stats;
```

23.1.7 GC_set_max_heap_size

```
function GC_set_max_heap_size
  input Integer size;
  output Boolean success;
end GC_set_max_heap_size;
```

23.1.8 addClassAnnotation

```
function addClassAnnotation
  input TypeName class_;
  input ExpressionOrModification annotate;
  output Boolean bool;
end addClassAnnotation;
```

23.1.9 addInitialState

```
function addInitialState
  input TypeName cl;
  input String state;
  input ExpressionOrModification annotate;
  output Boolean bool;
end addInitialState;
```

23.1.10 addTransition

```
function addTransition
  input TypeName cl;
  input String from;
  input String to;
  input String condition;
  input Boolean immediate = true;
  input Boolean reset = true;
  input Boolean synchronize = false;
  input Integer priority = 1;
  input ExpressionOrModification annotate;
  output Boolean bool;
end addTransition;
```

23.1.11 alarm

```
impure function alarm
  input Integer seconds;
  output Integer previousSeconds;
end alarm;
```

23.1.12 appendEnvironmentVar

```
Appends a variable to the environment variables list.
```

```
function appendEnvironmentVar
  input String var;
  input String value;
  output String result "returns \"error\" if the variable could not be appended";
end appendEnvironmentVar;
```

23.1.13 basename

```
function basename
  input String path;
  output String basename;
end basename;
```

23.1.14 buildEncryptedPackage

```
function buildEncryptedPackage
  input TypeName className "the class that should encrypted";
  input Boolean encrypt = true;
  output Boolean success;
end buildEncryptedPackage;
```

23.1.15 buildLabel

```
builds Lable.
```

```
function buildLabel
  input TypeName className "the class that should be built";
  input Real startTime = 0.0 "the start time of the simulation. <default> = 0.0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Integer numberOfIntervals = 500 "number of intervals in the result file.
↪<default> = 500";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default>␣
↪= 1e-6";
  input String method = "dassl" "integration method used for simulation. <default>␣
↪= dassl";
  input String fileNamePrefix = "" "fileNamePrefix. <default> = \"\"";
  input String options = "" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\
↪"";
  input String variableFilter = ".*" "Filter for variables that should store in␣
↪result file. <default> = \".*\"";
  input String cflags = "" "cflags. <default> = \"\"";
  input String simflags = "" "simflags. <default> = \"\"";
  output String[2] buildModelResults;
end buildLabel;
```

23.1.16 buildModel

```
builds a modelica model by generating c code and build it.
It does not run the code!
The only required argument is the className, while all others have some default␣
↪values.
simulate(className, [startTime], [stopTime], [numberOfIntervals], [tolerance],
 [method], [fileNamePrefix], [options], [outputFormat], [variableFilter], [cflags],
 [simflags])
Example command:
simulate(A);
```

```
function buildModel
  input TypeName className "the class that should be built";
  input Real startTime = "<default>" "the start time of the simulation. <default>␣
↪= 0.0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Real numberOfIntervals = 500 "number of intervals in the result file.
↪<default> = 500";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default>␣
↪= 1e-6";
  input String method = "<default>" "integration method used for simulation.
↪<default> = dassl";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"\"";
  input String options = "<default>" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\
↪"";
  input String variableFilter = ".*" "Filter for variables that should store in␣
↪result file. <default> = \".*\"";
  input String cflags = "<default>" "cflags. <default> = \"\"";
  input String simflags = "<default>" "simflags. <default> = \"\"";
  output String[2] buildModelResults;
end buildModel;
```

23.1.17 buildModelFMU

```
translates a modelica model into a Functional Mockup Unit.
The only required argument is the className, while all others have some default␣
↪values.
Example command:
buildModelFMU(className, version="2.0");
```

```
function buildModelFMU
  input TypeName className "the class that should translated";
  input String version = "2.0" "FMU version, 1.0 or 2.0.";
  input String fmuType = "me" "FMU type, me (model exchange), cs (co-simulation),␣
↪me_cs (both model exchange and co-simulation)";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \
↪"className\"";
  input String platforms[:] = {"static"} "The list of platforms to generate code␣
↪for. \"dynamic\"=current platform, dynamically link the runtime. \"static\
↪"=current platform, statically link everything. Else, use a host triple, e.g. \
↪"x86_64-linux-gnu\" or \"x86_64-w64-mingw32\"";
  input Boolean includeResources = false "include Modelica based resources via␣
↪loadResource or not";
  output String generatedFileName "Returns the full path of the generated FMU.";
end buildModelFMU;
```

23.1.18 buildOpenTURNSInterface

```
generates wrapper code for OpenTURNS
```

```
function buildOpenTURNSInterface
  input TypeName className;
  input String pythonTemplateFile;
  input Boolean showFlatModelica = false;
  output String outPythonScript;
end buildOpenTURNSInterface;
```

23.1.19 cd

```
change directory to the given path (which may be either relative or absolute)
returns the new working directory on success or a message on failure
if the given path is the empty string, the function simply returns the current␣
↪working directory.
```

```
function cd
  input String newWorkingDirectory = "";
  output String workingDirectory;
end cd;
```

23.1.20 checkAllModelsRecursive

```
Checks all models recursively and returns number of variables and equations.
```

```
function checkAllModelsRecursive
  input TypeName className;
  input Boolean checkProtected = false "Checks also protected classes if true";
  output String result;
end checkAllModelsRecursive;
```

23.1.21 checkCodeGraph

```
Checks if the given taskgraph has the same structure as the graph described in the␣
↪codefile.
```

```
function checkCodeGraph
  input String graphfile;
  input String codefile;
  output String[:] result;
end checkCodeGraph;
```

23.1.22 checkInterfaceOfPackages

```
function checkInterfaceOfPackages
  input TypeName cl;
  input String dependencyMatrix[:, :];
  output Boolean success;
end checkInterfaceOfPackages;
```

23.1.23 checkModel

```
Checks a model and returns number of variables and equations.
```

```
function checkModel
  input TypeName className;
  output String result;
end checkModel;
```

23.1.24 checkSettings

```
Display some diagnostics.
```

```
function checkSettings
  output CheckSettingsResult result;
end checkSettings;
```

23.1.25 checkTaskGraph

```
Checks if the given taskgraph has the same structure as the reference taskgraph␣
↪and if all attributes are set correctly.
```

```
function checkTaskGraph
  input String filename;
  input String reffilename;
  output String[:] result;
end checkTaskGraph;
```

23.1.26 classAnnotationExists

```
Check if annotation exists
```

```
function classAnnotationExists
  input TypeName className;
  input TypeName annotationName;
  output Boolean exists;
end classAnnotationExists;
```

23.1.27 clear

```
Clears everything: symboltable and variables.
```

```
function clear
  output Boolean success;
end clear;
```

23.1.28 clearCommandLineOptions

```
Resets all command-line flags to their default values.
```

```
function clearCommandLineOptions
  output Boolean success;
end clearCommandLineOptions;
```

23.1.29 clearDebugFlags

```
Resets all debug flags to their default values.
```

```
function clearDebugFlags
  output Boolean success;
end clearDebugFlags;
```

23.1.30 clearMessages

```
Clears the error buffer.
```

```
function clearMessages
  output Boolean success;
end clearMessages;
```

23.1.31 clearProgram

```
Clears loaded .
```

```
function clearProgram
  output Boolean success;
end clearProgram;
```

23.1.32 clearVariables

```
Clear all user defined variables.
```

```
function clearVariables
  output Boolean success;
end clearVariables;
```

23.1.33 closeSimulationResultFile

```
Closes the current simulation result file.
Only needed by Windows. Windows cannot handle reading and writing to the same file␣
↪from different processes.
To allow OMEdit to make successful simulation again on the same file we must close␣
↪the file after reading the Simulation Result Variables.
Even OMEdit only use this API for Windows.
```

```
function closeSimulationResultFile
  output Boolean success;
end closeSimulationResultFile;
```

23.1.34 codeToString

```
function codeToString
  input $Code className;
  output String string;
end codeToString;
```

23.1.35 compareFiles

```
impure function compareFiles
  input String file1;
  input String file2;
  output Boolean isEqual;
end compareFiles;
```

23.1.36 compareFilesAndMove

```
impure function compareFilesAndMove
  input String newFile;
  input String oldFile;
  output Boolean success;
end compareFilesAndMove;
```

23.1.37 compareSimulationResults

```
compares simulation results.
```

```
function compareSimulationResults
  input String filename;
  input String reffilename;
  input String logfilename;
  input Real relTol = 0.01;
  input Real absTol = 0.0001;
  input String[:] vars = fill("", 0);
  output String[:] result;
end compareSimulationResults;
```

23.1.38 convertUnits

```
function convertUnits
  input String s1;
  input String s2;
  output Boolean unitsCompatible;
  output Real scaleFactor;
  output Real offset;
end convertUnits;
```

23.1.39 copy

```
copies the source file to the destination file. Returns true if the file has been␣
↪copied.
```

```
function copy
  input String source;
  input String destination;
  output Boolean success;
end copy;
```

23.1.40 copyClass

```
Copies a class within the same level
```

```
function copyClass
  input TypeName className "the class that should be copied";
  input String newClassName "the name for new class";
  input TypeName withIn = $Code(TopLevel) "the with in path for new class";
  output Boolean result;
end copyClass;
```

23.1.41 countMessages

```
function countMessages
  output Integer numMessages;
  output Integer numErrors;
  output Integer numWarnings;
end countMessages;
```

23.1.42 deleteFile

```
Deletes a file with the given name.
```

```
function deleteFile
  input String fileName;
  output Boolean success;
end deleteFile;
```

23.1.43 deleteInitialState

```
function deleteInitialState
  input TypeName cl;
  input String state;
  output Boolean bool;
end deleteInitialState;
```

23.1.44 deleteTransition

```
function deleteTransition
  input TypeName cl;
  input String from;
  input String to;
  input String condition;
  input Boolean immediate;
  input Boolean reset;
  input Boolean synchronize;
  input Integer priority;
  output Boolean bool;
end deleteTransition;
```

23.1.45 deltaSimulationResults

```
calculates the sum of absolute errors.
```

```
function deltaSimulationResults
  input String filename;
  input String reffilename;
  input String method "method to compute then error. choose 1norm, 2norm, maxerr";
  input String[:] vars = fill("", 0);
  output Real result;
end deltaSimulationResults;
```

23.1.46 diffModelicaFileListings

```
Creates diffs of two strings corresponding to Modelica files
```

```
function diffModelicaFileListings
  input String before, after;
  input DiffFormat diffFormat = DiffFormat.color;
  output String result;
end diffModelicaFileListings;
```

23.1.47 diffSimulationResults

```
compares simulation results.
```

```
function diffSimulationResults
  input String actualFile;
  input String expectedFile;
  input String diffPrefix;
  input Real relTol = 1e-3 "y tolerance";
  input Real relTolDiffMinMax = 1e-4 "y tolerance based on the difference between␣
↪the maximum and minimum of the signal";
  input Real rangeDelta = 0.002 "x tolerance";
  input String[:] vars = fill("", 0);
  input Boolean keepEqualResults = false;
  output Boolean success;
  output String[:] failVars;
end diffSimulationResults;
```

23.1.48 diffSimulationResultsHtml

```
function diffSimulationResultsHtml
  input String var;
  input String actualFile;
  input String expectedFile;
  input Real relTol = 1e-3 "y tolerance";
  input Real relTolDiffMinMax = 1e-4 "y tolerance based on the difference between␣
↪the maximum and minimum of the signal";
```

(次のページに続く)

(前のページからの続き)

```
  input Real rangeDelta = 0.002 "x tolerance";
  output String html;
end diffSimulationResultsHtml;
```

23.1.49 directoryExists

```
function directoryExists
  input String dirName;
  output Boolean exists;
end directoryExists;
```

23.1.50 dirname

```
function dirname
  input String path;
  output String dirname;
end dirname;
```

23.1.51 disableNewInstantiation

```
function disableNewInstantiation
  output Boolean success;
end disableNewInstantiation;
```

23.1.52 dumpXMLDAE

```
Outputs the DAE system corresponding to a specific model.
```

```
function dumpXMLDAE
  input TypeName className;
  input String translationLevel = "flat" "flat, optimiser, backEnd, or stateSpace";
  input Boolean addOriginalIncidenceMatrix = false;
  input Boolean addSolvingInfo = false;
  input Boolean addMathMLCode = false;
  input Boolean dumpResiduals = false;
  input String fileNamePrefix = "<default>" "this is the className in string form␣
↪by default";
  input String rewriteRulesFile = "" "the file from where the rewiteRules are read,
 default is empty which means no rewrite rules";
  output Boolean success "if the function succeeded true/false";
  output String xmlfileName "the Xml file";
end dumpXMLDAE;
```

23.1.53 echo

```
echo(false) disables Interactive output, echo(true) enables it again.
```

```
function echo
  input Boolean setEcho;
  output Boolean newEcho;
end echo;
```

23.1.54 enableNewInstantiation

```
function enableNewInstantiation
  output Boolean success;
end enableNewInstantiation;
```

23.1.55 escapeXML

```
function escapeXML
  input String inStr;
  output String outStr;
end escapeXML;
```

23.1.56 exit

```
function exit
  input Integer status;
end exit;
```

23.1.57 exportToFigaro

```
function exportToFigaro
  input TypeName path;
  input String directory = cd();
  input String database;
  input String mode;
  input String options;
  input String processor;
  output Boolean success;
end exportToFigaro;
```

23.1.58 extendsFrom

```
returns true if the given class extends from the given base class
```

```
function extendsFrom
  input TypeName className;
  input TypeName baseClassName;
  output Boolean res;
end extendsFrom;
```

23.1.59 filterSimulationResults

```
function filterSimulationResults
  input String inFile;
  input String outFile;
  input String[:] vars;
  input Integer numberOfIntervals = 0 "0=Do not resample";
  input Boolean removeDescription = false;
  output Boolean success;
end filterSimulationResults;
```

23.1.60 generateCode

```
The input is a function name for which C-code is generated and compiled into a dll/
↪so
```

```
function generateCode
  input TypeName className;
  output Boolean success;
end generateCode;
```

23.1.61 generateEntryPoint

```
function generateEntryPoint
  input String fileName;
  input TypeName entryPoint;
  input String url = "https://trac.openmodelica.org/OpenModelica/newticket";
end generateEntryPoint;
```

23.1.62 generateHeader

```
function generateHeader
  input String fileName;
  output Boolean success;
end generateHeader;
```

23.1.63 generateJuliaHeader

```
function generateJuliaHeader
  input String fileName;
  output Boolean success;
end generateJuliaHeader;
```

23.1.64 generateScriptingAPI

```
function generateScriptingAPI
  input TypeName cl;
  input String name;
  output Boolean success;
  output String moFile;
  output String qtFile;
  output String qtHeader;
end generateScriptingAPI;
```

23.1.65 generateSeparateCode

```
function generateSeparateCode
  input TypeName className;
  input Boolean cleanCache = false "If true, the cache is reset between each␣
↪generated package. This conserves memory at the cost of speed.";
  output Boolean success;
end generateSeparateCode;
```

23.1.66 generateSeparateCodeDependencies

```
function generateSeparateCodeDependencies
  input String stampSuffix = ".c" "Suffix to add to dependencies (often .c.stamp)";
  output String[:] dependencies;
end generateSeparateCodeDependencies;
```

23.1.67 generateSeparateCodeDependenciesMakefile

```
function generateSeparateCodeDependenciesMakefile
  input String filename "The file to write the makefile to";
  input String directory = "" "The relative path of the generated files";
  input String suffix = ".c" "Often .stamp since we do not update all the files";
  output Boolean success;
end generateSeparateCodeDependenciesMakefile;
```

23.1.68 generateVerificationScenarios

```
function generateVerificationScenarios
  input TypeName path;
  output Boolean success;
end generateVerificationScenarios;
```

23.1.69 getAlgorithmCount

```
Counts the number of Algorithm sections in a class.
```

```
function getAlgorithmCount
  input TypeName class_;
  output Integer count;
end getAlgorithmCount;
```

23.1.70 getAlgorithmItemsCount

```
Counts the number of Algorithm items in a class.
```

```
function getAlgorithmItemsCount
  input TypeName class_;
  output Integer count;
end getAlgorithmItemsCount;
```

23.1.71 getAllSubtypeOf

```
Returns the list of all classes that extend from class_ given a parentClass where␣
↪the lookup for class_ should start
```

```
function getAllSubtypeOf
  input TypeName parentClass = $Code(AllLoadedClasses);
  input TypeName class_;
  input Boolean qualified = false;
  input Boolean includePartial = false;
  input Boolean sort = false;
```

(次のページに続く)

(前のページからの続き)

```
  output TypeName classNames[:];
end getAllSubtypeOf;
```

23.1.72 getAnnotationCount

```
Counts the number of Annotation sections in a class.
```

```
function getAnnotationCount
  input TypeName class_;
  output Integer count;
end getAnnotationCount;
```

23.1.73 getAnnotationModifierValue

```
function getAnnotationModifierValue
  input TypeName name;
  input String vendorannotation;
  input String modifiername;
  output String modifiernamevalue;
end getAnnotationModifierValue;
```

23.1.74 getAnnotationNamedModifiers

```
function getAnnotationNamedModifiers
  input TypeName name;
  input String vendorannotation;
  output String[:] modifiernamelist;
end getAnnotationNamedModifiers;
```

23.1.75 getAnnotationVersion

```
Returns the current annotation version.
```

```
function getAnnotationVersion
  output String annotationVersion;
end getAnnotationVersion;
```

23.1.76 getAstAsCorbaString

```
Print the whole AST on the CORBA format for records, e.g.
record Absyn.PROGRAM
classes = ...,
within_ = ...,
globalBuildTimes = ...
end Absyn.PROGRAM;
```

```
function getAstAsCorbaString
  input String fileName = "<interactive>";
  output String result "returns the string if fileName is interactive; else it␣
↪returns ok or error depending on if writing the file succeeded";
end getAstAsCorbaString;
```

23.1.77 getAvailableIndexReductionMethods

```
function getAvailableIndexReductionMethods
  output String[:] allChoices;
  output String[:] allComments;
end getAvailableIndexReductionMethods;
```

23.1.78 getAvailableLibraries

```
function getAvailableLibraries
  output String[:] libraries;
end getAvailableLibraries;
```

23.1.79 getAvailableMatchingAlgorithms

```
function getAvailableMatchingAlgorithms
  output String[:] allChoices;
  output String[:] allComments;
end getAvailableMatchingAlgorithms;
```

23.1.80 getAvailableTearingMethods

```
function getAvailableTearingMethods
  output String[:] allChoices;
  output String[:] allComments;
end getAvailableTearingMethods;
```

23.1.81 getBooleanClassAnnotation

```
Check if annotation exists and returns its value
```

```
function getBooleanClassAnnotation
  input TypeName className;
  input TypeName annotationName;
  output Boolean value;
end getBooleanClassAnnotation;
```

23.1.82 getBuiltinType

```
function getBuiltinType
  input TypeName cl;
  output String name;
end getBuiltinType;
```

23.1.83 getCFlags

```
CFLAGS
```

```
function getCFlags
  output String outString;
end getCFlags;
```

23.1.84 getCXXCompiler

```
CXX
```

```
function getCXXCompiler
  output String compiler;
end getCXXCompiler;
```

23.1.85 getClassComment

```
Returns the class comment.
```

```
function getClassComment
  input TypeName cl;
  output String comment;
end getClassComment;
```

23.1.86 getClassInformation

```
function getClassInformation
  input TypeName cl;
  output String restriction, comment;
  output Boolean partialPrefix, finalPrefix, encapsulatedPrefix;
  output String fileName;
  output Boolean fileReadOnly;
  output Integer lineNumberStart, columnNumberStart, lineNumberEnd,␣
↪columnNumberEnd;
  output String dimensions[:];
  output Boolean isProtectedClass;
  output Boolean isDocumentationClass;
  output String version;
  output String preferredView;
  output Boolean state;
  output String access;
end getClassInformation;
```

23.1.87 getClassNames

```
Returns the list of class names defined in the class.
```

```
function getClassNames
  input TypeName class_ = $Code(AllLoadedClasses);
  input Boolean recursive = false;
  input Boolean qualified = false;
  input Boolean sort = false;
  input Boolean builtin = false "List also builtin classes if true";
  input Boolean showProtected = false "List also protected classes if true";
  input Boolean includeConstants = false "List also constants in the class if true
↪";
  output TypeName classNames[:];
end getClassNames;
```

23.1.88 getClassRestriction

```
function getClassRestriction
  input TypeName cl;
  output String restriction;
end getClassRestriction;
```

23.1.89 getClassesInModelicaPath

```
MathCore-specific or not? Who knows!
```

```
function getClassesInModelicaPath
  output String classesInModelicaPath;
end getClassesInModelicaPath;
```

23.1.90 getCommandLineOptions

```
Returns all command line options who have non-default values as a list of
strings. The format of the strings is '--flag=value --flag2=value2'.
```

```
function getCommandLineOptions
  output String[:] flags;
end getCommandLineOptions;
```

23.1.91 getCompileCommand

```
function getCompileCommand
  output String compileCommand;
end getCompileCommand;
```

23.1.92 getCompiler

```
CC
```

```
function getCompiler
  output String compiler;
end getCompiler;
```

23.1.93 getComponentModifierNames

```
function getComponentModifierNames
  input TypeName class_;
  input String componentName;
  output String[:] modifiers;
end getComponentModifierNames;
```

23.1.94 getComponentModifierValue

```
function getComponentModifierValue
  input TypeName class_;
  input TypeName modifier;
  output String value;
end getComponentModifierValue;
```

23.1.95 getComponentModifierValues

```
function getComponentModifierValues
  input TypeName class_;
  input TypeName modifier;
  output String value;
end getComponentModifierValues;
```

23.1.96 getComponentsTest

```
function getComponentsTest
  input TypeName name;
  output Component[:] components;
  record Component
    String className;
    // when building record the constructor. Records are allowed to contain only␣
↪components of basic types, arrays of basic types or other records.
    String name;
    String comment;
    Boolean isProtected;
    Boolean isFinal;
    Boolean isFlow;
    Boolean isStream;
    Boolean isReplaceable;
    String variability "'constant', 'parameter', 'discrete', ''";
    String innerOuter "'inner', 'outer', ''";
    String inputOutput "'input', 'output', ''";
    String dimensions[:];
  end Component;
end getComponentsTest;
```

23.1.97 getConfigFlagValidOptions

```
Returns the list of valid options for a string config flag, and the description␣
↪strings for these options if available
```

```
function getConfigFlagValidOptions
  input String flag;
  output String validOptions[:];
  output String mainDescription;
  output String descriptions[:];
end getConfigFlagValidOptions;
```

23.1.98 getConnectionCount

```
Counts the number of connect equation in a class.
```

```
function getConnectionCount
  input TypeName className;
  output Integer count;
end getConnectionCount;
```

23.1.99 getDefaultOpenCLDevice

```
Returns the id for the default OpenCL device to be used.
```

```
function getDefaultOpenCLDevice
  output Integer defdevid;
end getDefaultOpenCLDevice;
```

23.1.100 getDerivedClassModifierNames

```
Returns the derived class modifier names.
Example command:
type Resistance = Real(final quantity="Resistance",final unit="Ohm");
getDerivedClassModifierNames(Resistance) => {"quantity","unit"}
```

```
function getDerivedClassModifierNames
  input TypeName className;
  output String[:] modifierNames;
end getDerivedClassModifierNames;
```

23.1.101 getDerivedClassModifierValue

```
Returns the derived class modifier value.
Example command:
type Resistance = Real(final quantity="Resistance",final unit="Ohm");
getDerivedClassModifierValue(Resistance, unit); => " = "Ohm""
getDerivedClassModifierValue(Resistance, quantity); => " = "Resistance""
```

```
function getDerivedClassModifierValue
  input TypeName className;
  input TypeName modifierName;
  output String modifierValue;
end getDerivedClassModifierValue;
```

23.1.102 getDerivedUnits

```
function getDerivedUnits
  input String baseUnit;
  output String[:] derivedUnits;
end getDerivedUnits;
```

23.1.103 getDocumentationAnnotation

```
Returns the documentaiton annotation defined in the class.
```

```
function getDocumentationAnnotation
  input TypeName cl;
  output String out[3] "{info,revision,infoHeader} TODO: Should be changed to have␣
↪2 outputs instead of an array of 2 Strings...";
end getDocumentationAnnotation;
```

23.1.104 getEnvironmentVar

```
Returns the value of the environment variable.
```

```
function getEnvironmentVar
  input String var;
  output String value "returns empty string on failure";
end getEnvironmentVar;
```

23.1.105 getEquationCount

```
Counts the number of Equation sections in a class.
```

```
function getEquationCount
  input TypeName class_;
  output Integer count;
end getEquationCount;
```

23.1.106 getEquationItemsCount

```
Counts the number of Equation items in a class.
```

```
function getEquationItemsCount
  input TypeName class_;
  output Integer count;
end getEquationItemsCount;
```

23.1.107 getErrorString

```
Returns the current error message. [file.mo:n:n-n:n:b] Error: message
```

```
impure function getErrorString
  input Boolean warningsAsErrors = false;
  output String errorString;
end getErrorString;
```

23.1.108 getImportCount

```
Counts the number of Import sections in a class.
```

```
function getImportCount
  input TypeName class_;
  output Integer count;
end getImportCount;
```

23.1.109 getIndexReductionMethod

```
function getIndexReductionMethod
  output String selected;
end getIndexReductionMethod;
```

23.1.110 getInheritedClasses

```
function getInheritedClasses
  input TypeName name;
  output TypeName inheritedClasses[:];
end getInheritedClasses;
```

23.1.111 getInitialAlgorithmCount

```
Counts the number of Initial Algorithm sections in a class.
```

```
function getInitialAlgorithmCount
  input TypeName class_;
  output Integer count;
end getInitialAlgorithmCount;
```

23.1.112 getInitialAlgorithmItemsCount

```
Counts the number of Initial Algorithm items in a class.
```

```
function getInitialAlgorithmItemsCount
  input TypeName class_;
  output Integer count;
end getInitialAlgorithmItemsCount;
```

23.1.113 getInitialEquationCount

```
Counts the number of Initial Equation sections in a class.
```

```
function getInitialEquationCount
  input TypeName class_;
  output Integer count;
end getInitialEquationCount;
```

23.1.114 getInitialEquationItemsCount

```
Counts the number of Initial Equation items in a class.
```

```
function getInitialEquationItemsCount
  input TypeName class_;
  output Integer count;
end getInitialEquationItemsCount;
```

23.1.115 getInitialStates

```
function getInitialStates
  input TypeName cl;
  output String[:, :] initialStates;
end getInitialStates;
```

23.1.116 getInstallationDirectoryPath

```
This returns OPENMODELICAHOME if it is set; on some platforms the default path is␣
↪returned if it is not set.
```

```
function getInstallationDirectoryPath
  output String installationDirectoryPath;
end getInstallationDirectoryPath;
```

23.1.117 getInstantiatedParametersAndValues

```
function getInstantiatedParametersAndValues
  input TypeName cls;
  output String[:] values;
end getInstantiatedParametersAndValues;
```

23.1.118 getLanguageStandard

```
Returns the current Modelica Language Standard in use.
```

```
function getLanguageStandard
  output String outVersion;
end getLanguageStandard;
```

23.1.119 getLinker

```
function getLinker
  output String linker;
end getLinker;
```

23.1.120 getLinkerFlags

```
function getLinkerFlags
  output String linkerFlags;
end getLinkerFlags;
```

23.1.121 getLoadedLibraries

```
function getLoadedLibraries
  output String[:, 2] libraries;
end getLoadedLibraries;
```

23.1.122 getMatchingAlgorithm

```
function getMatchingAlgorithm
  output String selected;
end getMatchingAlgorithm;
```

23.1.123 getMemorySize

```
function getMemorySize
  output Real memory(unit = "MiB");
end getMemorySize;
```

23.1.124 getMessagesString

```
see getErrorString()
```

```
function getMessagesString
  output String messagesString;
end getMessagesString;
```

23.1.125 getModelicaPath

```
Get the Modelica Library Path.
```

```
function getModelicaPath
  output String modelicaPath;
end getModelicaPath;
```

23.1.126 getNoSimplify

```
Returns true if noSimplify flag is set.
```

```
function getNoSimplify
  output Boolean noSimplify;
end getNoSimplify;
```

23.1.127 getNthAlgorithm

```
Returns the Nth Algorithm section.
```

```
function getNthAlgorithm
  input TypeName class_;
  input Integer index;
  output String result;
end getNthAlgorithm;
```

23.1.128 getNthAlgorithmItem

```
Returns the Nth Algorithm Item.
```

```
function getNthAlgorithmItem
  input TypeName class_;
  input Integer index;
  output String result;
end getNthAlgorithmItem;
```

23.1.129 getNthAnnotationString

```
Returns the Nth Annotation section as string.
```

```
function getNthAnnotationString
  input TypeName class_;
  input Integer index;
  output String result;
end getNthAnnotationString;
```

23.1.130 getNthConnection

```
Returns the Nth connection.
Example command:
getNthConnection(A) => {"from", "to", "comment"}
```

```
function getNthConnection
  input TypeName className;
  input Integer index;
  output String[:] result;
end getNthConnection;
```

23.1.131 getNthEquation

```
Returns the Nth Equation section.
```

```
function getNthEquation
  input TypeName class_;
  input Integer index;
  output String result;
end getNthEquation;
```

23.1.132 getNthEquationItem

```
Returns the Nth Equation Item.
```

```
function getNthEquationItem
  input TypeName class_;
  input Integer index;
  output String result;
end getNthEquationItem;
```

23.1.133 getNthImport

```
Returns the Nth Import as string.
```

```
function getNthImport
  input TypeName class_;
  input Integer index;
  output String out[3] "{\"Path\",\"Id\",\"Kind\"}";
end getNthImport;
```

23.1.134 getNthInitialAlgorithm

```
Returns the Nth Initial Algorithm section.
```

```
function getNthInitialAlgorithm
  input TypeName class_;
  input Integer index;
  output String result;
end getNthInitialAlgorithm;
```

23.1.135 getNthInitialAlgorithmItem

```
Returns the Nth Initial Algorithm Item.
```

```
function getNthInitialAlgorithmItem
  input TypeName class_;
  input Integer index;
  output String result;
end getNthInitialAlgorithmItem;
```

23.1.136 getNthInitialEquation

```
Returns the Nth Initial Equation section.
```

```
function getNthInitialEquation
  input TypeName class_;
  input Integer index;
  output String result;
end getNthInitialEquation;
```

23.1.137 getNthInitialEquationItem

```
Returns the Nth Initial Equation Item.
```

```
function getNthInitialEquationItem
  input TypeName class_;
  input Integer index;
  output String result;
end getNthInitialEquationItem;
```

23.1.138 getOrderConnections

```
Returns true if orderConnections flag is set.
```

```
function getOrderConnections
  output Boolean orderConnections;
end getOrderConnections;
```

23.1.139 getPackages

```
Returns the list of packages defined in the class.
```

```
function getPackages
  input TypeName class_ = $Code(AllLoadedClasses);
  output TypeName classNames[:];
end getPackages;
```

23.1.140 getParameterNames

```
function getParameterNames
  input TypeName class_;
  output String[:] parameters;
end getParameterNames;
```

23.1.141 getParameterValue

```
function getParameterValue
  input TypeName class_;
  input String parameterName;
  output String parameterValue;
end getParameterValue;
```

23.1.142 getSettings

```
function getSettings
  output String settings;
end getSettings;
```

23.1.143 getShowAnnotations

```
function getShowAnnotations
  output Boolean show;
end getShowAnnotations;
```

23.1.144 getSimulationOptions

```
function getSimulationOptions
  input TypeName name;
  input Real defaultStartTime = 0.0;
  input Real defaultStopTime = 1.0;
  input Real defaultTolerance = 1e-6;
  input Integer defaultNumberOfIntervals = 500 "May be overridden by defining␣
↪defaultInterval instead";
  input Real defaultInterval = 0.0 "If = 0.0, then numberOfIntervals is used to␣
↪calculate the step size";
  output Real startTime;
  output Real stopTime;
  output Real tolerance;
  output Integer numberOfIntervals;
  output Real interval;
end getSimulationOptions;
```

23.1.145 getSourceFile

```
Returns the filename of the class.
```

```
function getSourceFile
  input TypeName class_;
  output String filename "empty on failure";
end getSourceFile;
```

23.1.146 getTearingMethod

```
function getTearingMethod
  output String selected;
end getTearingMethod;
```

23.1.147 getTempDirectoryPath

```
Returns the current user temporary directory location.
```

```
function getTempDirectoryPath
  output String tempDirectoryPath;
end getTempDirectoryPath;
```

23.1.148 getTimeStamp

```
function getTimeStamp
  input TypeName cl;
  output Real timeStamp;
  output String timeStampAsString;
end getTimeStamp;
```

23.1.149 getTransitions

```
function getTransitions
  input TypeName cl;
  output String[:, :] transitions;
end getTransitions;
```

23.1.150 getUsedClassNames

```
Returns the list of class names used in the total program defined by the class.
```

```
function getUsedClassNames
  input TypeName className;
  output TypeName classNames[:];
end getUsedClassNames;
```

23.1.151 getUses

```
function getUses
  input TypeName pack;
  output String[:, :] uses;
end getUses;
```

23.1.152 getVectorizationLimit

```
function getVectorizationLimit
  output Integer vectorizationLimit;
end getVectorizationLimit;
```

23.1.153 getVersion

```
Returns the version of the Modelica compiler.
```

```
function getVersion
  input TypeName cl = $Code(OpenModelica);
  output String version;
end getVersion;
```

23.1.154 help

```
display the OpenModelica help text.
```

```
function help
  input String topic = "topics";
  output String helpText;
end help;
```

23.1.155 iconv

```
The iconv() function converts one multibyte characters from one character
set to another.
See man (3) iconv for more information.
```

```
function iconv
  input String string;
  input String from;
  input String to = "UTF-8";
  output String result;
end iconv;
```

23.1.156 importFMU

```
Imports the Functional Mockup Unit
Example command:
importFMU("A.fmu");
```

```
function importFMU
  input String filename "the fmu file name";
  input String workdir = "<default>" "The output directory for imported FMU files.
↪<default> will put the files to current working directory.";
  input Integer loglevel = 3 "loglevel_nothing=0;loglevel_fatal=1;loglevel_error=2;
↪loglevel_warning=3;loglevel_info=4;loglevel_verbose=5;loglevel_debug=6";
  input Boolean fullPath = false "When true the full output path is returned␣
↪otherwise only the file name.";
  input Boolean debugLogging = false "When true the FMU's debug output is printed.
↪";
  input Boolean generateInputConnectors = true "When true creates the input␣
↪connector pins.";
  input Boolean generateOutputConnectors = true "When true creates the output␣
↪connector pins.";
  output String generatedFileName "Returns the full path of the generated file.";
end importFMU;
```

23.1.157 importFMUModelDescription

```
Imports modelDescription.xml
Example command:
importFMUModelDescription("A.xml");
```

```
function importFMUModelDescription
  input String filename "the fmu file name";
  input String workdir = "<default>" "The output directory for imported FMU files.
↪<default> will put the files to current working directory.";
  input Integer loglevel = 3 "loglevel_nothing=0;loglevel_fatal=1;loglevel_error=2;
↪loglevel_warning=3;loglevel_info=4;loglevel_verbose=5;loglevel_debug=6";
  input Boolean fullPath = false "When true the full output path is returned␣
↪otherwise only the file name.";
  input Boolean debugLogging = false "When true the FMU's debug output is printed.
↪";
```

(次のページに続く)

(前のページからの続き)

```
  input Boolean generateInputConnectors = true "When true creates the input␣
↪connector pins.";
  input Boolean generateOutputConnectors = true "When true creates the output␣
↪connector pins.";
  output String generatedFileName "Returns the full path of the generated file.";
end importFMUModelDescription;
```

23.1.158 inferBindings

```
function inferBindings
  input TypeName path;
  output Boolean success;
end inferBindings;
```

23.1.159 instantiateModel

```
Instantiates the class and returns the flat Modelica code.
```

```
function instantiateModel
  input TypeName className;
  output String result;
end instantiateModel;
```

23.1.160 isBlock

```
function isBlock
  input TypeName cl;
  output Boolean b;
end isBlock;
```

23.1.161 isClass

```
function isClass
  input TypeName cl;
  output Boolean b;
end isClass;
```

23.1.162 isConnector

```
function isConnector
  input TypeName cl;
  output Boolean b;
end isConnector;
```

23.1.163 isEnumeration

```
function isEnumeration
  input TypeName cl;
  output Boolean b;
end isEnumeration;
```

23.1.164 isExperiment

```
function isExperiment
  input TypeName name;
  output Boolean res;
end isExperiment;
```

23.1.165 isFunction

```
function isFunction
  input TypeName cl;
  output Boolean b;
end isFunction;
```

23.1.166 isModel

```
function isModel
  input TypeName cl;
  output Boolean b;
end isModel;
```

23.1.167 isOperator

```
function isOperator
  input TypeName cl;
  output Boolean b;
end isOperator;
```

23.1.168 isOperatorFunction

```
function isOperatorFunction
  input TypeName cl;
  output Boolean b;
end isOperatorFunction;
```

23.1.169 isOperatorRecord

```
function isOperatorRecord
  input TypeName cl;
  output Boolean b;
end isOperatorRecord;
```

23.1.170 isOptimization

```
function isOptimization
  input TypeName cl;
  output Boolean b;
end isOptimization;
```

23.1.171 isPackage

```
function isPackage
  input TypeName cl;
  output Boolean b;
end isPackage;
```

23.1.172 isPartial

```
function isPartial
  input TypeName cl;
  output Boolean b;
end isPartial;
```

23.1.173 isProtectedClass

```
function isProtectedClass
  input TypeName cl;
  input String c2;
  output Boolean b;
end isProtectedClass;
```

23.1.174 isRecord

```
function isRecord
  input TypeName cl;
  output Boolean b;
end isRecord;
```

23.1.175 isShortDefinition

```
returns true if the definition is a short class definition
```

```
function isShortDefinition
  input TypeName class_;
  output Boolean isShortCls;
end isShortDefinition;
```

23.1.176 isType

```
function isType
  input TypeName cl;
  output Boolean b;
end isType;
```

23.1.177 linearize

```
creates a model with symbolic linearization matrixes
```

```
function linearize
  input TypeName className "the class that should simulated";
  input Real startTime = "<default>" "the start time of the simulation. <default>␣
↪= 0.0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Real numberOfIntervals = 500 "number of intervals in the result file.
↪<default> = 500";
  input Real stepSize = 0.002 "step size that is used for the result file.
↪<default> = 0.002";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default>␣
↪= 1e-6";
  input String method = "<default>" "integration method used for simulation.
↪<default> = dassl";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"\"";
  input Boolean storeInTemp = false "storeInTemp. <default> = false";
  input Boolean noClean = false "noClean. <default> = false";
  input String options = "<default>" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\
↪"";
  input String variableFilter = ".*" "Filter for variables that should store in␣
↪result file. <default> = \".*\"";
  input String cflags = "<default>" "cflags. <default> = \"\"";
```

(次のページに続く)

(前のページからの続き)

```
  input String simflags = "<default>" "simflags. <default> = \"\"";
  output String linearizationResult;
end linearize;
```

23.1.178 list

```
Lists the contents of the given class, or all loaded classes.
```

```
function list
  input TypeName class_ = $Code(AllLoadedClasses);
  input Boolean interfaceOnly = false;
  input Boolean shortOnly = false "only short class definitions";
  input ExportKind exportKind = ExportKind.Absyn;
  output String contents;
end list;
```

23.1.179 listFile

```
Lists the contents of the file given by the class.
```

```
function listFile
  input TypeName class_;
  input Boolean nestedClasses = true;
  output String contents;
end listFile;
```

23.1.180 listVariables

```
Lists the names of the active variables in the scripting environment.
```

```
function listVariables
  output TypeName variables[:];
end listVariables;
```

23.1.181 loadEncryptedPackage

```
function loadEncryptedPackage
  input String fileName;
  input String workdir = "<default>" "The output directory for imported encrypted␣
↪files. <default> will put the files to current working directory.";
  output Boolean success;
end loadEncryptedPackage;
```

23.1.182 loadFile

```
load file (*.mo) and merge it with the loaded AST.
```

```
function loadFile
  input String fileName;
  input String encoding = "UTF-8";
  input Boolean uses = true;
  output Boolean success;
end loadFile;
```

23.1.183 loadFileInteractive

```
function loadFileInteractive
  input String filename;
  input String encoding = "UTF-8";
  output TypeName names[:];
end loadFileInteractive;
```

23.1.184 loadFileInteractiveQualified

```
function loadFileInteractiveQualified
  input String filename;
  input String encoding = "UTF-8";
  output TypeName names[:];
end loadFileInteractiveQualified;
```

23.1.185 loadFiles

```
load files (*.mo) and merges them with the loaded AST.
```

```
function loadFiles
  input String[:] fileNames;
  input String encoding = "UTF-8";
  input Integer numThreads = OpenModelica.Scripting.numProcessors();
  output Boolean success;
end loadFiles;
```

23.1.186 loadModel

```
Loads the Modelica Standard Library.
```

```
function loadModel
  input TypeName className;
  input String[:] priorityVersion = {"default"};
  input Boolean notify = false "Give a notification of the libraries and versions␣
↪that were loaded";
  input String languageStandard = "" "Override the set language standard. Parse␣
↪with the given setting, but do not change it permanently.";
  input Boolean requireExactVersion = false "If the version is required to be␣
↪exact, if there is a uses Modelica(version=\"3.2\"), Modelica 3.2.1 will not␣
↪match it.";
  output Boolean success;
end loadModel;
```

23.1.187 loadModelica3D

```
function loadModelica3D
  input String version = "3.2.1";
  output Boolean status;
end loadModelica3D;
```

23.1.188 loadOMSimulator

```
loads the OMSimulator DLL from default path
```

```
function loadOMSimulator
  output Integer status;
end loadOMSimulator;
```

23.1.189 loadString

```
Parses the data and merges the resulting AST with ithe
loaded AST.
If a filename is given, it is used to provide error-messages as if the string
was read in binary format from a file with the same name.
The file is converted to UTF-8 from the given character set.
When merge is true the classes cNew in the file will be merged with the already␣
↪loaded classes cOld in the following way:
1. get all the inner class definitions from cOld that were loaded from a different␣
↪file than itself
2. append all elements from step 1 to class cNew public list
NOTE: Encoding is deprecated as *ALL* strings are now UTF-8 encoded.
```

```
function loadString
  input String data;
  input String filename = "<interactive>";
  input String encoding = "UTF-8";
  input Boolean merge = false "if merge is true the parsed AST is merged with the␣
↪existing AST, default to false which means that is replaced, not merged";
  output Boolean success;
end loadString;
```

23.1.190 mkdir

```
create directory of given path (which may be either relative or absolute)
returns true if directory was created or already exists.
```

```
function mkdir
  input String newDirectory;
  output Boolean success;
end mkdir;
```

23.1.191 moveClass

```
Moves a class up or down depending on the given offset, where a positive
offset moves the class down and a negative offset up. The offset is truncated
if the resulting index is outside the class list. It retains the visibility of
the class by adding public/protected sections when needed, and merges sections
of the same type if the class is moved from a section it was alone in. Returns
true if the move was successful, otherwise false.
```

```
function moveClass
  input TypeName className "the class that should be moved";
  input Integer offset "Offset in the class list.";
  output Boolean result;
end moveClass;
```

23.1.192 moveClassToBottom

```
Moves a class to the bottom of its enclosing class. Returns true if the move
was successful, otherwise false.
```

```
function moveClassToBottom
  input TypeName className;
  output Boolean result;
end moveClassToBottom;
```

23.1.193 moveClassToTop

```
Moves a class to the top of its enclosing class. Returns true if the move
was successful, otherwise false.
```

```
function moveClassToTop
  input TypeName className;
  output Boolean result;
end moveClassToTop;
```

23.1.194 ngspicetoModelica

```
Converts ngspice netlist to Modelica code. Modelica file is created in the same␣
↪directory as netlist file.
```

```
function ngspicetoModelica
  input String netlistfileName;
  output Boolean success = false;
end ngspicetoModelica;
```

23.1.195 numProcessors

```
function numProcessors
  output Integer result;
end numProcessors;
```

23.1.196 oms_RunFile

```
function oms_RunFile
  input String filename;
  output Integer status;
end oms_RunFile;
```

23.1.197 oms_addBus

```
function oms_addBus
  input String cref;
  output Integer status;
end oms_addBus;
```

23.1.198 oms_addConnection

```
function oms_addConnection
  input String crefA;
  input String crefB;
  output Integer status;
end oms_addConnection;
```

23.1.199 oms_addConnector

```
function oms_addConnector
  input String cref;
  input oms_causality causality;
  input oms_signal_type type_;
  output Integer status;
end oms_addConnector;
```

23.1.200 oms_addConnectorToBus

```
function oms_addConnectorToBus
  input String busCref;
  input String connectorCref;
  output Integer status;
end oms_addConnectorToBus;
```

23.1.201 oms_addConnectorToTLMBus

```
function oms_addConnectorToTLMBus
  input String busCref;
  input String connectorCref;
  input String type_;
  output Integer status;
end oms_addConnectorToTLMBus;
```

23.1.202 oms_addDynamicValueIndicator

```
function oms_addDynamicValueIndicator
  input String signal;
  input String lower;
  input String upper;
  input Real stepSize;
  output Integer status;
end oms_addDynamicValueIndicator;
```

23.1.203 oms_addEventIndicator

```
function oms_addEventIndicator
  input String signal;
  output Integer status;
end oms_addEventIndicator;
```

23.1.204 oms_addExternalModel

```
function oms_addExternalModel
  input String cref;
  input String path;
  input String startscript;
  output Integer status;
end oms_addExternalModel;
```

23.1.205 oms_addSignalsToResults

```
function oms_addSignalsToResults
  input String cref;
  input String regex;
  output Integer status;
end oms_addSignalsToResults;
```

23.1.206 oms_addStaticValueIndicator

```
function oms_addStaticValueIndicator
  input String signal;
  input Real lower;
  input Real upper;
  input Real stepSize;
  output Integer status;
end oms_addStaticValueIndicator;
```

23.1.207 oms_addSubModel

```
function oms_addSubModel
  input String cref;
  input String fmuPath;
  output Integer status;
end oms_addSubModel;
```

23.1.208 oms_addSystem

```
function oms_addSystem
  input String cref;
  input oms_system type_;
  output Integer status;
end oms_addSystem;
```

23.1.209 oms_addTLMBus

```
function oms_addTLMBus
  input String cref;
  input oms_tlm_domain domain;
  input Integer dimensions;
  input oms_tlm_interpolation interpolation;
  output Integer status;
end oms_addTLMBus;
```

23.1.210 oms_addTLMConnection

```
function oms_addTLMConnection
  input String crefA;
  input String crefB;
  input Real delay;
  input Real alpha;
  input Real linearimpedance;
  input Real angularimpedance;
  output Integer status;
end oms_addTLMConnection;
```

23.1.211 oms_addTimeIndicator

```
function oms_addTimeIndicator
  input String signal;
  output Integer status;
end oms_addTimeIndicator;
```

23.1.212 oms_cancelSimulation_asynchronous

```
function oms_cancelSimulation_asynchronous
  input String cref;
  output Integer status;
end oms_cancelSimulation_asynchronous;
```

23.1.213 oms_compareSimulationResults

```
function oms_compareSimulationResults
  input String filenameA;
  input String filenameB;
  input String var;
  input Real relTol;
  input Real absTol;
  output Integer status;
end oms_compareSimulationResults;
```

23.1.214 oms_copySystem

```
function oms_copySystem
  input String source;
  input String target;
  output Integer status;
end oms_copySystem;
```

23.1.215 oms_delete

```
function oms_delete
  input String cref;
  output Integer status;
end oms_delete;
```

23.1.216 oms_deleteConnection

```
function oms_deleteConnection
  input String crefA;
  input String crefB;
  output Integer status;
end oms_deleteConnection;
```

23.1.217 oms_deleteConnectorFromBus

```
function oms_deleteConnectorFromBus
  input String busCref;
  input String connectorCref;
  output Integer status;
end oms_deleteConnectorFromBus;
```

23.1.218 oms_deleteConnectorFromTLMBus

```
function oms_deleteConnectorFromTLMBus
  input String busCref;
  input String connectorCref;
  output Integer status;
end oms_deleteConnectorFromTLMBus;
```

23.1.219 oms_export

```
function oms_export
  input String cref;
  input String filename;
  output Integer status;
end oms_export;
```

23.1.220 oms_exportDependencyGraphs

```
function oms_exportDependencyGraphs
  input String cref;
  input String initialization;
  input String simulation;
  output Integer status;
end oms_exportDependencyGraphs;
```

23.1.221 oms_extractFMIKind

```
function oms_extractFMIKind
  input String filename;
  output Integer status;
  output Integer kind;
end oms_extractFMIKind;
```

23.1.222 oms_faultInjection

```
function oms_faultInjection
  input String signal;
  input oms_fault_type_enu_t faultType;
  input Real faultValue;
  output Integer status;
end oms_faultInjection;
```

23.1.223 oms_getBoolean

```
function oms_getBoolean
  input String cref;
  output Integer status;
  output Boolean value;
end oms_getBoolean;
```

23.1.224 oms_getFixedStepSize

```
function oms_getFixedStepSize
  input String cref;
  output Integer status;
  output Real stepSize;
end oms_getFixedStepSize;
```

23.1.225 oms_getInteger

```
function oms_getInteger
  input String cref;
  output Integer status;
  input Integer value;
end oms_getInteger;
```

23.1.226 oms_getModelState

```
function oms_getModelState
  input String cref;
  output Integer status;
  output Integer modelState;
end oms_getModelState;
```

23.1.227 oms_getReal

```
function oms_getReal
  input String cref;
  output Integer status;
  output Real value;
end oms_getReal;
```

23.1.228 oms_getSolver

```
function oms_getSolver
  input String cref;
  output Integer status;
  output Integer solver;
end oms_getSolver;
```

23.1.229 oms_getStartTime

```
function oms_getStartTime
  input String cref;
  output Integer status;
  output Real startTime;
end oms_getStartTime;
```

23.1.230 oms_getStopTime

```
function oms_getStopTime
  input String cref;
  output Integer status;
  output Real stopTime;
end oms_getStopTime;
```

23.1.231 oms_getSubModelPath

```
function oms_getSubModelPath
  input String cref;
  output Integer status;
  output String path;
end oms_getSubModelPath;
```

23.1.232 oms_getSystemType

```
function oms_getSystemType
  input String cref;
  output Integer status;
  output Integer type_;
end oms_getSystemType;
```

23.1.233 oms_getTolerance

```
function oms_getTolerance
  input String cref;
  output Integer status;
  output Real absoluteTolerance;
  output Real relativeTolerance;
end oms_getTolerance;
```

23.1.234 oms_getVariableStepSize

```
function oms_getVariableStepSize
  input String cref;
  output Integer status;
  output Real initialStepSize;
  output Real minimumStepSize;
  output Real maximumStepSize;
end oms_getVariableStepSize;
```

23.1.235 oms_getVersion

```
Returns the version of the OMSimulator.
```

```
function oms_getVersion
  output String version;
end oms_getVersion;
```

23.1.236 oms_importFile

```
function oms_importFile
  input String filename;
  output Integer status;
  output String cref;
end oms_importFile;
```

23.1.237 oms_initialize

```
function oms_initialize
  input String cref;
  output Integer status;
end oms_initialize;
```

23.1.238 oms_instantiate

```
function oms_instantiate
  input String cref;
  output Integer status;
end oms_instantiate;
```

23.1.239 oms_list

```
function oms_list
  input String cref;
  output Integer status;
  output String contents;
end oms_list;
```

23.1.240 oms_listUnconnectedConnectors

```
function oms_listUnconnectedConnectors
  input String cref;
  output Integer status;
  output String contents;
end oms_listUnconnectedConnectors;
```

23.1.241 oms_loadSnapshot

```
function oms_loadSnapshot
  input String cref;
  input String snapshot;
  output Integer status;
end oms_loadSnapshot;
```

23.1.242 oms_newModel

```
function oms_newModel
  input String cref;
  output Integer status;
end oms_newModel;
```

23.1.243 oms_parseModelName

```
function oms_parseModelName
  input String contents;
  output Integer status;
  output String cref;
end oms_parseModelName;
```

23.1.244 oms_removeSignalsFromResults

```
function oms_removeSignalsFromResults
  input String cref;
  input String regex;
  output Integer status;
end oms_removeSignalsFromResults;
```

23.1.245 oms_rename

```
function oms_rename
  input String cref;
  input String newCref;
  output Integer status;
end oms_rename;
```

23.1.246 oms_reset

```
function oms_reset
  input String cref;
  output Integer status;
end oms_reset;
```

23.1.247 oms_setBoolean

```
function oms_setBoolean
  input String cref;
  input Boolean value;
  output Integer status;
end oms_setBoolean;
```

23.1.248 oms_setCommandLineOption

```
function oms_setCommandLineOption
  input String cmd;
  output Integer status;
end oms_setCommandLineOption;
```

23.1.249 oms_setFixedStepSize

```
function oms_setFixedStepSize
  input String cref;
  input Real stepSize;
  output Integer status;
end oms_setFixedStepSize;
```

23.1.250 oms_setInteger

```
function oms_setInteger
  input String cref;
  input Integer value;
  output Integer status;
end oms_setInteger;
```

23.1.251 oms_setLogFile

```
function oms_setLogFile
  input String filename;
  output Integer status;
end oms_setLogFile;
```

23.1.252 oms_setLoggingInterval

```
function oms_setLoggingInterval
  input String cref;
  input Real loggingInterval;
  output Integer status;
end oms_setLoggingInterval;
```

23.1.253 oms_setLoggingLevel

```
function oms_setLoggingLevel
  input Integer logLevel;
  output Integer status;
end oms_setLoggingLevel;
```

23.1.254 oms_setReal

```
function oms_setReal
  input String cref;
  input Real value;
  output Integer status;
end oms_setReal;
```

23.1.255 oms_setRealInputDerivative

```
function oms_setRealInputDerivative
  input String cref;
  input Real value;
  output Integer status;
end oms_setRealInputDerivative;
```

23.1.256 oms_setResultFile

```
function oms_setResultFile
  input String cref;
  input String filename;
  input Integer bufferSize;
  output Integer status;
end oms_setResultFile;
```

23.1.257 oms_setSignalFilter

```
function oms_setSignalFilter
  input String cref;
  input String regex;
  output Integer status;
end oms_setSignalFilter;
```

23.1.258 oms_setSolver

```
function oms_setSolver
  input String cref;
  input oms_solver solver;
  output Integer status;
end oms_setSolver;
```

23.1.259 oms_setStartTime

```
function oms_setStartTime
  input String cref;
  input Real startTime;
  output Integer status;
end oms_setStartTime;
```

23.1.260 oms_setStopTime

```
function oms_setStopTime
  input String cref;
  input Real stopTime;
  output Integer status;
end oms_setStopTime;
```

23.1.261 oms_setTLMPositionAndOrientation

```
function oms_setTLMPositionAndOrientation
  input String cref;
  input Real x1;
  input Real x2;
  input Real x3;
  input Real A11;
  input Real A12;
  input Real A13;
  input Real A21;
  input Real A22;
  input Real A23;
  input Real A31;
  input Real A32;
  input Real A33;
  output Integer status;
end oms_setTLMPositionAndOrientation;
```

23.1.262 oms_setTLMSocketData

```
function oms_setTLMSocketData
  input String cref;
  input String address;
  input Integer managerPort;
  input Integer monitorPort;
  output Integer status;
end oms_setTLMSocketData;
```

23.1.263 oms_setTempDirectory

```
function oms_setTempDirectory
  input String newTempDir;
  output Integer status;
end oms_setTempDirectory;
```

23.1.264 oms_setTolerance

```
function oms_setTolerance
  input String cref;
  input Real absoluteTolerance;
  input Real relativeTolerance;
  output Integer status;
end oms_setTolerance;
```

23.1.265 oms_setVariableStepSize

```
function oms_setVariableStepSize
  input String cref;
  input Real initialStepSize;
  input Real minimumStepSize;
  input Real maximumStepSize;
  output Integer status;
end oms_setVariableStepSize;
```

23.1.266 oms_setWorkingDirectory

```
function oms_setWorkingDirectory
  input String newWorkingDir;
  output Integer status;
end oms_setWorkingDirectory;
```

23.1.267 oms_simulate

```
function oms_simulate
  input String cref;
  output Integer status;
end oms_simulate;
```

23.1.268 oms_stepUntil

```
function oms_stepUntil
  input String cref;
  input Real stopTime;
  output Integer status;
end oms_stepUntil;
```

23.1.269 oms_terminate

```
function oms_terminate
  input String cref;
  output Integer status;
end oms_terminate;
```

23.1.270 optimize

```
optimize a modelica/optimica model by generating c code, build it and run the␣
↪optimization executable.
The only required argument is the className, while all others have some default␣
↪values.
simulate(className, [startTime], [stopTime], [numberOfIntervals], [stepSize],␣
↪[tolerance], [fileNamePrefix], [options], [outputFormat], [variableFilter],␣
↪[cflags], [simflags])
Example command:
simulate(A);
```

```
function optimize
  input TypeName className "the class that should simulated";
  input Real startTime = "<default>" "the start time of the simulation. <default>␣
↪= 0.0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Real numberOfIntervals = 500 "number of intervals in the result file.
↪<default> = 500";
  input Real stepSize = 0.002 "step size that is used for the result file.
↪<default> = 0.002";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default>␣
↪= 1e-6";
  input String method = DAE.SCONST("optimization") "optimize a modelica/optimica␣
↪model.";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"\"";
  input Boolean storeInTemp = false "storeInTemp. <default> = false";
  input Boolean noClean = false "noClean. <default> = false";
```

(次のページに続く)

(前のページからの続き)

```
  input String options = "<default>" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\
↪"";
  input String variableFilter = ".*" "Filter for variables that should store in␣
↪result file. <default> = \".*\"";
  input String cflags = "<default>" "cflags. <default> = \"\"";
  input String simflags = "<default>" "simflags. <default> = \"\"";
  output String optimizationResults;
end optimize;
```

23.1.271 parseEncryptedPackage

```
function parseEncryptedPackage
  input String fileName;
  input String workdir = "<default>" "The output directory for imported encrypted␣
↪files. <default> will put the files to current working directory.";
  output TypeName names[:];
end parseEncryptedPackage;
```

23.1.272 parseFile

```
function parseFile
  input String filename;
  input String encoding = "UTF-8";
  output TypeName names[:];
end parseFile;
```

23.1.273 parseString

```
function parseString
  input String data;
  input String filename = "<interactive>";
  output TypeName names[:];
end parseString;
```

23.1.274 plot

```
Launches a plot window using OMPlot.
```

```
function plot
  input VariableNames vars "The variables you want to plot";
  input Boolean externalWindow = false "Opens the plot in a new plot window";
  input String fileName = "<default>" "The filename containing the variables.
↪<default> will read the last simulation result";
  input String title = "" "This text will be used as the diagram title.";
  input String grid = "detailed" "Sets the grid for the plot i.e simple, detailed,␣
↪none.";
```

(次のページに続く)

(前のページからの続き)

```
  input Boolean logX = false "Determines whether or not the horizontal axis is␣
↪logarithmically scaled.";
  input Boolean logY = false "Determines whether or not the vertical axis is␣
↪logarithmically scaled.";
  input String xLabel = "time" "This text will be used as the horizontal label in␣
↪the diagram.";
  input String yLabel = "" "This text will be used as the vertical label in the␣
↪diagram.";
  input Real xRange[2] = {0.0, 0.0} "Determines the horizontal interval that is␣
↪visible in the diagram. {0,0} will select a suitable range.";
  input Real yRange[2] = {0.0, 0.0} "Determines the vertical interval that is␣
↪visible in the diagram. {0,0} will select a suitable range.";
  input Real curveWidth = 1.0 "Sets the width of the curve.";
  input Integer curveStyle = 1 "Sets the style of the curve. SolidLine=1,␣
↪DashLine=2, DotLine=3, DashDotLine=4, DashDotDotLine=5, Sticks=6, Steps=7.";
  input String legendPosition = "top" "Sets the POSITION of the legend i.e left,␣
↪right, top, bottom, none.";
  input String footer = "" "This text will be used as the diagram footer.";
  input Boolean autoScale = true "Use auto scale while plotting.";
  input Boolean forceOMPlot = false "if true launches OMPlot and doesn't call␣
↪callback function even if it is defined.";
  output Boolean success "Returns true on success";
end plot;
```

23.1.275 plotAll

```
Works in the same way as plot(), but does not accept any
variable names as input. Instead, all variables are part of the plot window.
Example command sequences:
simulate(A);plotAll();
simulate(A);plotAll(externalWindow=true);
simulate(A,fileNamePrefix="B");simulate(C);plotAll(x,fileName="B.mat");
```

```
function plotAll
  input Boolean externalWindow = false "Opens the plot in a new plot window";
  input String fileName = "<default>" "The filename containing the variables.
↪<default> will read the last simulation result";
  input String title = "" "This text will be used as the diagram title.";
  input String grid = "detailed" "Sets the grid for the plot i.e simple, detailed,␣
↪none.";
  input Boolean logX = false "Determines whether or not the horizontal axis is␣
↪logarithmically scaled.";
  input Boolean logY = false "Determines whether or not the vertical axis is␣
↪logarithmically scaled.";
  input String xLabel = "time" "This text will be used as the horizontal label in␣
↪the diagram.";
  input String yLabel = "" "This text will be used as the vertical label in the␣
↪diagram.";
  input Real xRange[2] = {0.0, 0.0} "Determines the horizontal interval that is␣
↪visible in the diagram. {0,0} will select a suitable range.";
  input Real yRange[2] = {0.0, 0.0} "Determines the vertical interval that is␣
↪visible in the diagram. {0,0} will select a suitable range.";
  input Real curveWidth = 1.0 "Sets the width of the curve.";
  input Integer curveStyle = 1 "Sets the style of the curve. SolidLine=1,␣
↪DashLine=2, DotLine=3, DashDotLine=4, DashDotDotLine=5, Sticks=6, Steps=7.";
  input String legendPosition = "top" "Sets the POSITION of the legend i.e left,␣
↪right, top, bottom, none.";
```

(次のページに続く)

(前のページからの続き)

```
  input String footer = "" "This text will be used as the diagram footer.";
  input Boolean autoScale = true "Use auto scale while plotting.";
  input Boolean forceOMPlot = false "if true launches OMPlot and doesn't call␣
↪callback function even if it is defined.";
  output Boolean success "Returns true on success";
end plotAll;
```

23.1.276 plotParametric

```
Launches a plotParametric window using OMPlot. Returns true on success.
Example command sequences:
simulate(A);plotParametric(x,y);
simulate(A);plotParametric(x,y, externalWindow=true);
```

```
function plotParametric
  input VariableName xVariable;
  input VariableName yVariable;
  input Boolean externalWindow = false "Opens the plot in a new plot window";
  input String fileName = "<default>" "The filename containing the variables.
↪<default> will read the last simulation result";
  input String title = "" "This text will be used as the diagram title.";
  input String grid = "detailed" "Sets the grid for the plot i.e simple, detailed,␣
↪none.";
  input Boolean logX = false "Determines whether or not the horizontal axis is␣
↪logarithmically scaled.";
  input Boolean logY = false "Determines whether or not the vertical axis is␣
↪logarithmically scaled.";
  input String xLabel = "time" "This text will be used as the horizontal label in␣
↪the diagram.";
  input String yLabel = "" "This text will be used as the vertical label in the␣
↪diagram.";
  input Real xRange[2] = {0.0, 0.0} "Determines the horizontal interval that is␣
↪visible in the diagram. {0,0} will select a suitable range.";
  input Real yRange[2] = {0.0, 0.0} "Determines the vertical interval that is␣
↪visible in the diagram. {0,0} will select a suitable range.";
  input Real curveWidth = 1.0 "Sets the width of the curve.";
  input Integer curveStyle = 1 "Sets the style of the curve. SolidLine=1,␣
↪DashLine=2, DotLine=3, DashDotLine=4, DashDotDotLine=5, Sticks=6, Steps=7.";
  input String legendPosition = "top" "Sets the POSITION of the legend i.e left,␣
↪right, top, bottom, none.";
  input String footer = "" "This text will be used as the diagram footer.";
  input Boolean autoScale = true "Use auto scale while plotting.";
  input Boolean forceOMPlot = false "if true launches OMPlot and doesn't call␣
↪callback function even if it is defined.";
  output Boolean success "Returns true on success";
end plotParametric;
```

23.1.277 readFile

```
The contents of the given file are returned.
Note that if the function fails, the error message is returned as a string instead␣
↪of multiple output or similar.
```

```
impure function readFile
  input String fileName;
  output String contents;
end readFile;
```

23.1.278 readFileNoNumeric

```
Returns the contents of the file, with anything resembling a (real) number␣
↪stripped out, and at the end adding:
Filter count from number domain: n.
This should probably be changed to multiple outputs; the filtered string and an␣
↪integer.
Does anyone use this API call?
```

```
function readFileNoNumeric
  input String fileName;
  output String contents;
end readFileNoNumeric;
```

23.1.279 readSimulationResult

```
Reads a result file, returning a matrix corresponding to the variables and size␣
↪given.
```

```
function readSimulationResult
  input String filename;
  input VariableNames variables;
  input Integer size = 0 "0=read any size... If the size is not the same as the␣
↪result-file, this function fails";
  output Real result[:, :];
end readSimulationResult;
```

23.1.280 readSimulationResultSize

```
The number of intervals that are present in the output file.
```

```
function readSimulationResultSize
  input String fileName;
  output Integer sz;
end readSimulationResultSize;
```

23.1.281 readSimulationResultVars

```
Returns the variables in the simulation file; you can use val() and plot()␣
↪commands using these names.
```

```
function readSimulationResultVars
  input String fileName;
  input Boolean readParameters = true;
  input Boolean openmodelicaStyle = false;
  output String[:] vars;
end readSimulationResultVars;
```

23.1.282 realpath

```
Get full path name of file or directory name
```

```
function realpath
  input String name "Absolute or relative file or directory name";
  output String fullName "Full path of 'name'";
end realpath;
```

23.1.283 reduceTerms

```
reduce terms.
```

```
function reduceTerms
  input TypeName className "the class that should be built";
  input Real startTime = 0.0 "the start time of the simulation. <default> = 0.0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Integer numberOfIntervals = 500 "number of intervals in the result file.
↪<default> = 500";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default>␣
↪= 1e-6";
  input String method = "dassl" "integration method used for simulation. <default>␣
↪= dassl";
  input String fileNamePrefix = "" "fileNamePrefix. <default> = \"\"";
  input String options = "" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\
↪"";
  input String variableFilter = ".*" "Filter for variables that should store in␣
↪result file. <default> = \".*\"";
  input String cflags = "" "cflags. <default> = \"\"";
  input String simflags = "" "simflags. <default> = \"\"";
  input String labelstoCancel = "";
  output String[2] buildModelResults;
end reduceTerms;
```

23.1.284 regex

```
Sets the error buffer and returns -1 if the regex does not compile.
The returned result is the same as POSIX regex():
The first value is the complete matched string
The rest are the substrings that you wanted.
For example:
regex(lorem," \([A-Za-z]*\) \([A-Za-z]*\) ",maxMatches=3)
=> {" ipsum dolor ","ipsum","dolor"}
This means if you have n groups, you want maxMatches=n+1
```

```
function regex
  input String str;
  input String re;
  input Integer maxMatches = 1 "The maximum number of matches that will be returned
↪";
  input Boolean extended = true "Use POSIX extended or regular syntax";
  input Boolean caseInsensitive = false;
  output Integer numMatches "-1 is an error, 0 means no match, else returns a␣
↪number 1..maxMatches";
  output String matchedSubstrings[maxMatches] "unmatched strings are returned as␣
↪empty";
end regex;
```

23.1.285 regexBool

```
Returns true if the string matches the regular expression.
```

```
function regexBool
  input String str;
  input String re;
  input Boolean extended = true "Use POSIX extended or regular syntax";
  input Boolean caseInsensitive = false;
  output Boolean matches;
end regexBool;
```

23.1.286 regularFileExists

```
function regularFileExists
  input String fileName;
  output Boolean exists;
end regularFileExists;
```

23.1.287 reloadClass

```
reloads the file associated with the given (loaded class)
```

```
function reloadClass
  input TypeName name;
  input String encoding = "UTF-8";
  output Boolean success;
end reloadClass;
```

23.1.288 remove

```
removes a file or directory of given path (which may be either relative or␣
↪absolute).
```

```
function remove
  input String path;
  output Boolean success "Returns true on success.";
end remove;
```

23.1.289 removeComponentModifiers

```
function removeComponentModifiers
  input TypeName class_;
  input String componentName;
  input Boolean keepRedeclares = false;
  output Boolean success;
end removeComponentModifiers;
```

23.1.290 removeExtendsModifiers

```
function removeExtendsModifiers
  input TypeName className;
  input TypeName baseClassName;
  input Boolean keepRedeclares = false;
  output Boolean success;
end removeExtendsModifiers;
```

23.1.291 reopenStandardStream

```
function reopenStandardStream
  input StandardStream _stream;
  input String filename;
  output Boolean success;
end reopenStandardStream;
```

23.1.292 rewriteBlockCall

```
Function for property modeling, transforms block calls into instantiations for a␣
↪loaded model
```

```
function rewriteBlockCall
  input TypeName className;
  input TypeName inDefs;
  output Boolean success;
end rewriteBlockCall;
```

23.1.293 runOpenTURNSPythonScript

```
runs OpenTURNS with the given python script returning the log file
```

```
function runOpenTURNSPythonScript
  input String pythonScriptFile;
  output String logOutputFile;
end runOpenTURNSPythonScript;
```

23.1.294 runScript

```
Runs the mos-script specified by the filename.
```

```
impure function runScript
  input String fileName "*.mos";
  output String result;
end runScript;
```

23.1.295 runScriptParallel

```
function runScriptParallel
  input String scripts[:];
  input Integer numThreads = numProcessors();
  input Boolean useThreads = false;
  output Boolean results[:];
end runScriptParallel;
```

23.1.296 save

```
function save
  input TypeName className;
  output Boolean success;
end save;
```

23.1.297 saveAll

```
save the entire loaded AST to file.
```

```
function saveAll
  input String fileName;
  output Boolean success;
end saveAll;
```

23.1.298 saveModel

```
function saveModel
  input String fileName;
  input TypeName className;
  output Boolean success;
end saveModel;
```

23.1.299 saveTotalModel

```
Save the className model in a single file, together with all
the other classes that it depends upon, directly and indirectly.
This file can be later reloaded with the loadFile() API function,
which loads className and all the other needed classes into memory.
This is useful to allow third parties to run a certain model (e.g. for debugging)
without worrying about all the library dependencies.
Please note that SaveTotal file is not a valid Modelica .mo file according to the
specification, and cannot be loaded in OMEdit - it can only be loaded with␣
↪loadFile().
```

```
function saveTotalModel
  input String fileName;
  input TypeName className;
  input Boolean stripAnnotations = false;
  input Boolean stripComments = false;
  output Boolean success;
end saveTotalModel;
```

23.1.300 saveTotalSCode

23.1.301 searchClassNames

```
Searches for the class name in the all the loaded classes.
Example command:
searchClassNames("ground");
searchClassNames("ground", true);
```

```
function searchClassNames
  input String searchText;
  input Boolean findInText = false;
  output TypeName classNames[:];
end searchClassNames;
```

23.1.302 setAnnotationVersion

```
Sets the annotation version.
```

```
function setAnnotationVersion
  input String annotationVersion;
  output Boolean success;
end setAnnotationVersion;
```

23.1.303 setCFlags

```
CFLAGS
```

```
function setCFlags
  input String inString;
  output Boolean success;
end setCFlags;
```

23.1.304 setCXXCompiler

```
CXX
```

```
function setCXXCompiler
  input String compiler;
  output Boolean success;
end setCXXCompiler;
```

23.1.305 setCheapMatchingAlgorithm

```
example input: 3
```

```
function setCheapMatchingAlgorithm
  input Integer matchingAlgorithm;
  output Boolean success;
end setCheapMatchingAlgorithm;
```

23.1.306 setClassComment

```
Sets the class comment.
```

```
function setClassComment
  input TypeName class_;
  input String filename;
  output Boolean success;
end setClassComment;
```

23.1.307 setCommandLineOptions

```
The input is a regular command-line flag given to OMC, e.g. -d=failtrace or -
↪g=MetaModelica
```

```
function setCommandLineOptions
  input String option;
  output Boolean success;
end setCommandLineOptions;
```

23.1.308 setCompileCommand

```
function setCompileCommand
  input String compileCommand;
  output Boolean success;
end setCompileCommand;
```

23.1.309 setCompiler

```
CC
```

```
function setCompiler
  input String compiler;
  output Boolean success;
end setCompiler;
```

23.1.310 setCompilerFlags

```
function setCompilerFlags
  input String compilerFlags;
  output Boolean success;
end setCompilerFlags;
```

23.1.311 setCompilerPath

```
function setCompilerPath
  input String compilerPath;
  output Boolean success;
end setCompilerPath;
```

23.1.312 setDebugFlags

```
example input: failtrace,-noevalfunc
```

```
function setDebugFlags
  input String debugFlags;
  output Boolean success;
end setDebugFlags;
```

23.1.313 setDefaultOpenCLDevice

```
Sets the default OpenCL device to be used.
```

```
function setDefaultOpenCLDevice
  input Integer defdevid;
  output Boolean success;
end setDefaultOpenCLDevice;
```

23.1.314 setDocumentationAnnotation

```
function setDocumentationAnnotation
  input TypeName class_;
  input String info = "";
  input String revisions = "";
  output Boolean bool;
end setDocumentationAnnotation;
```

23.1.315 setEnvironmentVar

```
function setEnvironmentVar
  input String var;
  input String value;
  output Boolean success;
end setEnvironmentVar;
```

23.1.316 setIndexReductionMethod

```
example input: dynamicStateSelection
```

```
function setIndexReductionMethod
  input String method;
  output Boolean success;
end setIndexReductionMethod;
```

23.1.317 setInitXmlStartValue

```
function setInitXmlStartValue
  input String fileName;
  input String variableName;
  input String startValue;
  input String outputFile;
  output Boolean success = false;
end setInitXmlStartValue;
```

23.1.318 setInstallationDirectoryPath

```
Sets the OPENMODELICAHOME environment variable. Use this method instead of␣
↪setEnvironmentVar.
```

```
function setInstallationDirectoryPath
  input String installationDirectoryPath;
  output Boolean success;
end setInstallationDirectoryPath;
```

23.1.319 setLanguageStandard

```
Sets the Modelica Language Standard.
```

```
function setLanguageStandard
  input String inVersion;
  output Boolean success;
end setLanguageStandard;
```

23.1.320 setLinker

```
function setLinker
  input String linker;
  output Boolean success;
end setLinker;
```

23.1.321 setLinkerFlags

```
function setLinkerFlags
  input String linkerFlags;
  output Boolean success;
end setLinkerFlags;
```

23.1.322 setMatchingAlgorithm

```
example input: omc
```

```
function setMatchingAlgorithm
  input String matchingAlgorithm;
  output Boolean success;
end setMatchingAlgorithm;
```

23.1.323 setModelicaPath

```
The Modelica Library Path - MODELICAPATH in the language specification;␣
↪OPENMODELICALIBRARY in OpenModelica.
```

```
function setModelicaPath
  input String modelicaPath;
  output Boolean success;
end setModelicaPath;
```

23.1.324 setNoSimplify

```
Sets the noSimplify flag.
```

```
function setNoSimplify
  input Boolean noSimplify;
  output Boolean success;
end setNoSimplify;
```

23.1.325 setOrderConnections

```
Sets the orderConnection flag.
```

```
function setOrderConnections
  input Boolean orderConnections;
  output Boolean success;
end setOrderConnections;
```

23.1.326 setPlotCommand

```
function setPlotCommand
  input String plotCommand;
  output Boolean success;
end setPlotCommand;
```

23.1.327 setPostOptModules

```
example input: lateInline,inlineArrayEqn,removeSimpleEquations.
```

```
function setPostOptModules
  input String modules;
  output Boolean success;
end setPostOptModules;
```

23.1.328 setPreOptModules

```
example input: removeFinalParameters,removeSimpleEquations,expandDerOperator
```

```
function setPreOptModules
  input String modules;
  output Boolean success;
end setPreOptModules;
```

23.1.329 setShowAnnotations

```
function setShowAnnotations
  input Boolean show;
  output Boolean success;
end setShowAnnotations;
```

23.1.330 setSourceFile

```
function setSourceFile
  input TypeName class_;
  input String filename;
  output Boolean success;
end setSourceFile;
```

23.1.331 setTearingMethod

```
example input: omcTearing
```

```
function setTearingMethod
  input String tearingMethod;
  output Boolean success;
end setTearingMethod;
```

23.1.332 setTempDirectoryPath

```
function setTempDirectoryPath
  input String tempDirectoryPath;
  output Boolean success;
end setTempDirectoryPath;
```

23.1.333 setVectorizationLimit

```
function setVectorizationLimit
  input Integer vectorizationLimit;
  output Boolean success;
end setVectorizationLimit;
```

23.1.334 simulate

```
simulates a modelica model by generating c code, build it and run the simulation␣
↪executable.
The only required argument is the className, while all others have some default␣
↪values.
simulate(className, [startTime], [stopTime], [numberOfIntervals], [tolerance],
 [method], [fileNamePrefix], [options], [outputFormat], [variableFilter], [cflags],
 [simflags])
Example command:
simulate(A);
```

```
function simulate
  input TypeName className "the class that should simulated";
  input Real startTime = "<default>" "the start time of the simulation. <default>␣
↪= 0.0";
  input Real stopTime = 1.0 "the stop time of the simulation. <default> = 1.0";
  input Real numberOfIntervals = 500 "number of intervals in the result file.
↪<default> = 500";
  input Real tolerance = 1e-6 "tolerance used by the integration method. <default>␣
↪= 1e-6";
  input String method = "<default>" "integration method used for simulation.
↪<default> = dassl";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \"\"";
  input String options = "<default>" "options. <default> = \"\"";
  input String outputFormat = "mat" "Format for the result file. <default> = \"mat\
↪"";
  input String variableFilter = ".*" "Filter for variables that should store in␣
↪result file. <default> = \".*\"";
  input String cflags = "<default>" "cflags. <default> = \"\"";
  input String simflags = "<default>" "simflags. <default> = \"\"";
  output SimulationResult simulationResults;
  record SimulationResult
    String resultFile;
    String simulationOptions;
    String messages;
    Real timeFrontend;
    Real timeBackend;
    Real timeSimCode;
    Real timeTemplates;
    Real timeCompile;
    Real timeSimulation;
    Real timeTotal;
  end SimulationResult;
end simulate;
```

23.1.335 solveLinearSystem

```
Solve A*X = B, using dgesv or lp_solve (if any variable in X is integer)
Returns for solver dgesv: info>0: Singular for element i. info<0: Bad input.
For solver lp_solve: ???
```

```
function solveLinearSystem
  input Real[size(B, 1), size(B, 1)] A;
  input Real[:] B;
  input LinearSystemSolver solver = LinearSystemSolver.dgesv;
  input Integer[:] isInt = {-1} "list of indices that are integers";
  output Real[size(B, 1)] X;
  output Integer info;
end solveLinearSystem;
```

23.1.336 sortStrings

```
function sortStrings
  input String arr[:];
  output String sorted[:];
end sortStrings;
```

23.1.337 stat

```
impure function stat
  input String fileName;
  output Boolean success;
  output Real fileSize;
  output Real mtime;
end stat;
```

23.1.338 stringReplace

```
function stringReplace
  input String str;
  input String source;
  input String target;
  output String res;
end stringReplace;
```

23.1.339 stringSplit

```
Splits the string at the places given by the character
```

```
function stringSplit
  input String string;
  input String token "single character only";
  output String[:] strings;
end stringSplit;
```

23.1.340 stringTypeName

```
function stringTypeName
  input String str;
  output TypeName cl;
end stringTypeName;
```

23.1.341 stringVariableName

```
function stringVariableName
  input String str;
  output VariableName cl;
end stringVariableName;
```

23.1.342 strtok

```
Splits the strings at the places given by the token, for example:
strtok("abcbdef","b") => {"a","c","def"}
strtok("abcbdef","cd") => {"ab","ef"}
```

```
function strtok
  input String string;
  input String token;
  output String[:] strings;
end strtok;
```

23.1.343 system

```
Similar to system(3). Executes the given command in the system shell.
```

```
impure function system
  input String callStr "String to call: sh -c $callStr";
  input String outputFile = "" "The output is redirected to this file (unless␣
↪already done by callStr)";
  output Integer retval "Return value of the system call; usually 0 on success";
end system;
```

23.1.344 system_parallel

```
Similar to system(3). Executes the given commands in the system shell, in parallel␣
↪if omc was compiled using OpenMP.
```

```
impure function system_parallel
  input String callStr[:] "String to call: sh -c $callStr";
  input Integer numThreads = numProcessors();
  output Integer retval[:] "Return value of the system call; usually 0 on success";
end system_parallel;
```

23.1.345 testsuiteFriendlyName

```
function testsuiteFriendlyName
  input String path;
  output String fixed;
end testsuiteFriendlyName;
```

23.1.346 threadWorkFailed

23.1.347 translateGraphics

```
function translateGraphics
  input TypeName className;
  output String result;
end translateGraphics;
```

23.1.348 translateModelFMU

```
translates a modelica model into a Functional Mockup Unit.
The only required argument is the className, while all others have some default␣
↪values.
Example command:
translateModelFMU(className, version="2.0");
```

```
function translateModelFMU
  input TypeName className "the class that should translated";
  input String version = "2.0" "FMU version, 1.0 or 2.0.";
  input String fmuType = "me" "FMU type, me (model exchange), cs (co-simulation),␣
↪me_cs (both model exchange and co-simulation)";
  input String fileNamePrefix = "<default>" "fileNamePrefix. <default> = \
↪"className\"";
  input Boolean includeResources = false "include Modelica based resources via␣
↪loadResource or not";
  output String generatedFileName "Returns the full path of the generated FMU.";
end translateModelFMU;
```

23.1.349 typeNameString

```
function typeNameString
  input TypeName cl;
  output String out;
end typeNameString;
```

23.1.350 typeNameStrings

```
function typeNameStrings
  input TypeName cl;
  output String out[:];
end typeNameStrings;
```

23.1.351 typeOf

```
function typeOf
  input VariableName variableName;
  output String result;
end typeOf;
```

23.1.352 unloadOMSimulator

```
free the OMSimulator instances
```

```
function unloadOMSimulator
  output Integer status;
end unloadOMSimulator;
```

23.1.353 updateConnection

```
function updateConnection
  input TypeName className;
  input String from;
  input String to;
  input ExpressionOrModification annotate;
  output Boolean result;
end updateConnection;
```

23.1.354 updateConnectionNames

```
function updateConnectionNames
  input TypeName className;
  input String from;
  input String to;
  input String fromNew;
  input String toNew;
  output Boolean result;
end updateConnectionNames;
```

23.1.355 updateInitialState

```
function updateInitialState
  input TypeName cl;
  input String state;
  input ExpressionOrModification annotate;
  output Boolean bool;
end updateInitialState;
```

23.1.356 updateTransition

```
function updateTransition
  input TypeName cl;
  input String from;
  input String to;
  input String oldCondition;
  input Boolean oldImmediate;
  input Boolean oldReset;
  input Boolean oldSynchronize;
  input Integer oldPriority;
  input String newCondition;
  input Boolean newImmediate;
  input Boolean newReset;
  input Boolean newSynchronize;
  input Integer newPriority;
  input ExpressionOrModification annotate;
  output Boolean bool;
end updateTransition;
```

23.1.357 uriToFilename

```
function uriToFilename
  input String uri;
  output String filename = "";
end uriToFilename;
```

23.1.358 val

```
Return the value of a variable at a given time in the simulation results
```

```
function val
  input VariableName var;
  input Real timePoint = 0.0;
  input String fileName = "<default>" "The contents of the currentSimulationResult␣
↪variable";
  output Real valAtTime;
end val;
```

23.1.359 verifyCompiler

```
function verifyCompiler
  output Boolean compilerWorks;
end verifyCompiler;
```

23.1.360 writeFile

```
Write the data to file. Returns true on success.
```

```
impure function writeFile
  input String fileName;
  input String data;
  input Boolean append = false;
  output Boolean success;
end writeFile;
```

23.2 Simulation Parameter Sweep

Following example shows how to update the parameters and re-run the simulation without compiling the model.

```
loadFile("BouncingBall.mo");
getErrorString();
// build the model once
buildModel(BouncingBall);
getErrorString();
for i in 1:3 loop
  // We update the parameter e start value from 0.7 to "0.7 + i".
  value := 0.7 + i;
  // call the generated simulation code to produce a result file BouncingBall%i%_
↪res.mat
  system("./BouncingBall -override=e="+String(value)+" -r=BouncingBall" +␣
↪String(i) + "_res.mat");
  getErrorString();
end for;
```

We used the BouncingBall.mo in the example above. The above example produces three result files each containing different start value for *e* i.e., 1.7, 2.7, 3.7.

23.3 Examples

The following is an interactive session with the OpenModelica environment including some of the abovementioned commands and examples. First we start the system, and use the command line interface from OMShell, OMNotebook, or command window of some of the other tools.

We type in a very small model:

```
model Test "Testing OpenModelica Scripts"
  Real x, y;
equation
  x = 5.0+time; y = 6.0;
end Test;
```

We give the command to flatten a model:

```
>>> instantiateModel(Test)
class Test "Testing OpenModelica Scripts"
  Real x;
  Real y;
equation
  x = 5.0 + time;
  y = 6.0;
end Test;
```

A range expression is typed in:

```
>>> a:=1:10
{1,2,3,4,5,6,7,8,9,10}
```

It is multiplied by 2:

```
>>> a*2
{2,4,6,8,10,12,14,16,18,20}
```

The variables are cleared:

```
>>> clearVariables()
true
```

We print the loaded class test from its internal representation:

```
>>> list(Test)
model Test "Testing OpenModelica Scripts"
  Real x, y;
equation
  x = 5.0 + time;
  y = 6.0;
end Test;
```

We get the name and other properties of a class:

```
>>> getClassNames()
{Test,ProfilingTest}
>>> getClassComment(Test)
"Testing OpenModelica Scripts"
>>> isPartial(Test)
false
>>> isPackage(Test)
false
>>> isModel(Test)
true
>>> checkModel(Test)
"Check of Test completed successfully.
Class Test has 2 equation(s) and 2 variable(s).
2 of these are trivial equation(s)."
```

The common combination of a simulation followed by getting a value and doing a plot:

```
>>> simulate(Test, stopTime=3.0)
record SimulationResult
    resultFile = "<<DOCHOME>>/Test_res.mat",
    simulationOptions = "startTime = 0.0, stopTime = 3.0, numberOfIntervals = 500,␣
↪tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'Test', options = '',␣
↪outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''",
    messages = "LOG_SUCCESS       | info    | The initialization finished␣
↪successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
stdout            | info    | Time measurements are stored in Test_prof.html␣
↪(human-readable) and Test_prof.xml (for XSL transforms or more details)
",
    timeFrontend = 0.005024900000000001,
    timeBackend = 0.0105711,
    timeSimCode = 0.0008427,
    timeTemplates = 0.010635,
    timeCompile = 2.6113352,
    timeSimulation = 0.4413759,
    timeTotal = 3.082195
end SimulationResult;
>>> val(x , 2.0)
7.0
```

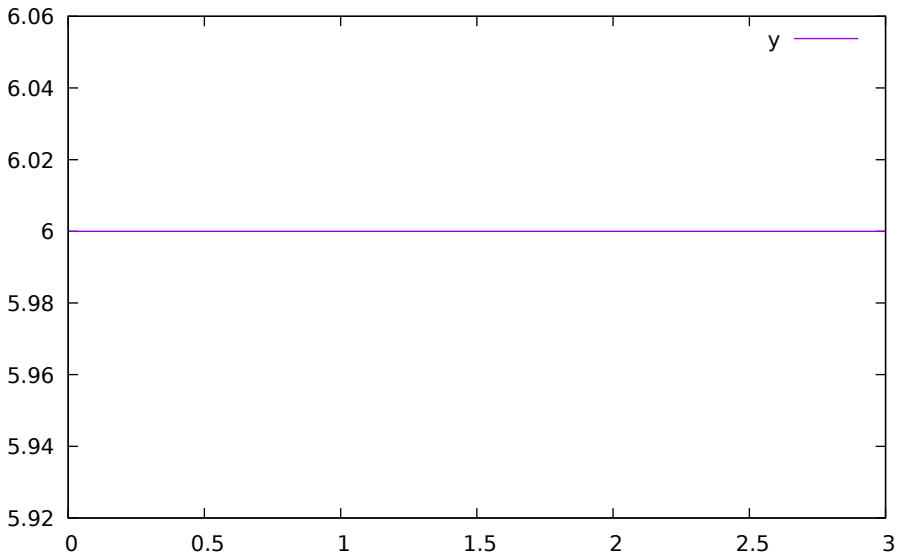


Figure 23.1: Plot generated by OpenModelica+gnuplot

```
>>> plotall()
```

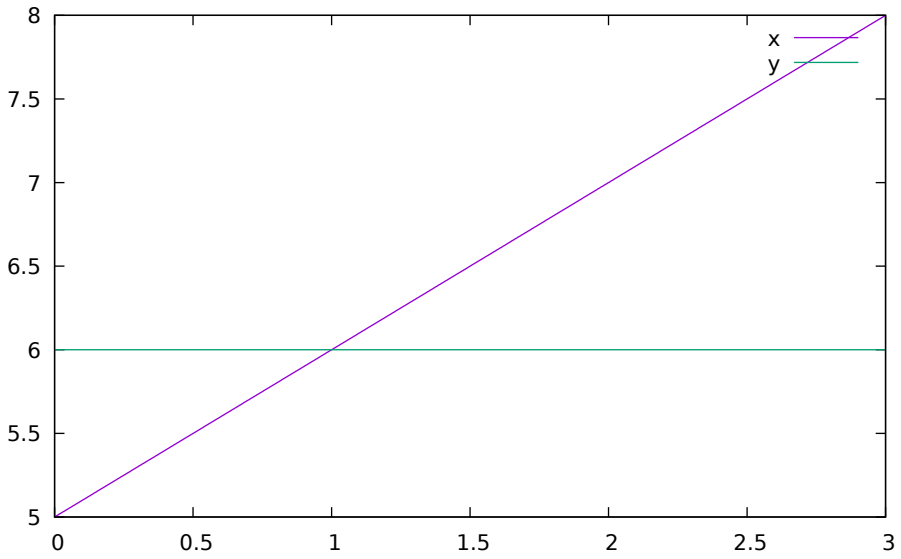


Figure 23.2: Plot generated by OpenModelica+gnuplot

23.3.1 Interactive Function Calls, Reading, and Writing

We enter an assignment of a vector expression, created by the range construction expression 1:12, to be stored in the variable x. The type and the value of the expression is returned.

```
>>> x := 1:12
{1,2,3,4,5,6,7,8,9,10,11,12}
```

The function bubblesort is called to sort this vector in descending order. The sorted result is returned together with its type. Note that the result vector is of type Real[:], instantiated as Real[12], since this is the declared type of the function result. The input Integer vector was automatically converted to a Real vector according to the Modelica type coercion rules.

```
>>> loadFile(getInstallationDirectoryPath() + "/share/doc/omc/testmodels/
↪bubblesort.mo")
true
>>> bubblesort(x)
{12.0,11.0,10.0,9.0,8.0,7.0,6.0,5.0,4.0,3.0,2.0,1.0}
```

Now we want to try another small application, a simplex algorithm for optimization. First read in a small matrix containing coefficients that define a simplex problem to be solved:

```
>>> a := {
  {-1,-1,-1, 0, 0, 0, 0, 0, 0},
  {-1, 1, 0, 1, 0, 0, 0, 0, 5},
  { 1, 4, 0, 0, 1, 0, 0, 0, 45},
  { 2, 1, 0, 0, 0, 1, 0, 0, 27},
  { 3,-4, 0, 0, 0, 0, 1, 0, 24},
  { 0, 0, 1, 0, 0, 0, 0, 1, 4}
}
{{-1,-1,-1,0,0,0,0,0,0},{-1,1,0,1,0,0,0,0,5},{1,4,0,0,1,0,0,0,45},{2,1,0,0,0,1,0,0,
↪27},{3,-4,0,0,0,0,1,0,24},{0,0,1,0,0,0,0,1,4}}
```

```
function pivot1
  input Real b[:,:];
  input Integer p;
  input Integer q;
  output Real a[size(b,1),size(b,2)];
protected
  Integer M;
  Integer N;
algorithm
  a := b;
  N := size(a,1)-1;
  M := size(a,2)-1;
  for j in 1:N loop
    for k in 1:M loop
      if j<>p and k<>q then
       a[j,k] := a[j,k]-0.3*j;
      end if;
    end for;
  end for;
  a[p,q] := 0.05;
end pivot1;

function misc_simplex1
  input Real matr[:,:];
  output Real x[size(matr,2)-1];
  output Real z;
  output  Integer q;
  output  Integer p;
protected
  Real a[size(matr,1),size(matr,2)];
  Integer M;
  Integer N;
algorithm
  N := size(a,1)-1;
  M := size(a,2)-1;
  a := matr;
  p:=0;q:=0;
  a := pivot1(a,p+1,q+1);
  while not (q==(M) or p==(N)) loop
    q := 0;
    while not (q == (M) or a[0+1,q+1]>1) loop
      q:=q+1;
    end while;
    p := 0;
    while not (p == (N) or a[p+1,q+1]>0.1) loop
      p:=p+1;
    end while;
    if (q < M) and (p < N) and(p>0) and (q>0) then
      a := pivot1(a,p,q);
    end if;
  if(p<=0) and (q<=0) then
     a := pivot1(a,p+1,q+1);
  end if;
  if(p<=0) and (q>0) then
     a := pivot1(a,p+1,q);
  end if;
  if(p>0) and (q<=0) then
     a := pivot1(a,p,q+1);
  end if;
  end while;
  z := a[1,M];
  x := {a[1,i] for i in 1:size(x,1)};
```

(次のページに続く)

(前のページからの続き)

```
  for i in 1:10 loop
   for j in 1:M loop
    x[j] := x[j]+x[j]*0.01;
   end for;
  end for;
end misc_simplex1;
```

Then call the simplex algorithm implemented as the Modelica function simplex1. This function returns four results, which are represented as a tuple of four return values:

```
>>> misc_simplex1(a)
({0.05523110627056022,-1.104622125411205,-1.104622125411205,0.0,0.0,0.0,0.0,0.0},0.
↪0,8,1)
```

第24章　OpenModelica Compiler Flags

Usage: omc [Options] (Model.mo | Script.mos) [Libraries | .mo-files]

- Libraries: Fully qualified names of libraries to load before processing Model or Script. The libraries should be separated by spaces: Lib1 Lib2 ... LibN.

24.1 Options

-d, --debug

Sets debug flags. Use *--help=debug* to see available flags.

String list (default *empty*).

-h, --help

Displays the help text. Use --help=topics for more information.

String (default *empty*).

--v, --version

Print the version and exit.

Boolean (default `false`).

--target

Sets the target compiler to use.

String (default gcc). Valid options:

- gcc
- msvc
- msvc10
- msvc12
- msvc13
- msvc15
- msvc19
- vxworks69

- debugrt

-g, --grammar

Sets the grammar and semantics to accept.

String (default Modelica). Valid options:

- Modelica
- MetaModelica
- ParModelica
- Optimica
- PDEModelica

--annotationVersion

Sets the annotation version that should be used.

String (default 3.x). Valid options:

- 1.x
- 2.x
- 3.x

--std

Sets the language standard that should be used.

String (default latest). Valid options:

- 1.x
- 2.x
- 3.1
- 3.2
- 3.3
- latest

--showErrorMessages

Show error messages immediately when they happen.

Boolean (default `false`).

--showAnnotations

Show annotations in the flattened code.

Boolean (default `false`).

--noSimplify

Do not simplify expressions if set.

Boolean (default `false`).

--preOptModules

Sets the pre optimization modules to use in the back end. See *--help=optmodules* for more info.

String list (default normalInlineFunction,evaluateParameters,simplifyIfEquations,expandDerOperator,clockPartitioning,findStateOr
Valid options:

- introduceOutputAliases (Introduces aliases for top-level outputs.)
- clockPartitioning (Does the clock partitioning.)
- collapseArrayExpressions (Simplifies {x[1],x[2],x[3]} → x for arrays of whole variable references (simplifies code generation).)
- comSubExp (Introduces alias assignments for variables which are assigned to simple terms i.e. a = b/c; d = b/c; --> a=d)
- dumpDAE (dumps the DAE representation of the current transformation state)
- dumpDAEXML (dumps the DAE as xml representation of the current transformation state)
- encapsulateWhenConditions (This module replaces each when condition with a boolean variable.)
- evalFunc (evaluates functions partially)
- evaluateParameters (Evaluates parameters with annotation(Evaluate=true). Use '--evaluateFinalParameters=true' or '--evaluateProtectedParameters=true' to specify additional parameters to be evaluated. Use '--replaceEvaluatedParameters=true' if the evaluated parameters should be replaced in the DAE. To evaluate all parameters in the Frontend use -d=evaluateAllParameters.)
- expandDerOperator (Expands der(expr) using Derive.differentiteExpTime.)
- findStateOrder (Sets derivative information to states.)
- inlineArrayEqn (This module expands all array equations to scalar equations.)
- normalInlineFunction (Perform function inlining for function with annotation Inline=true.)
- inputDerivativesForDynOpt (Allowed derivatives of inputs in dyn. optimization.)
- introduceDerAlias (Adds for every der-call an alias equation e.g. dx = der(x).)
- removeEqualRHS (Detects equal expressions of the form a=<exp> and b=<exp> and substitutes them to get speed up.)
- removeProtectedParameters (Replace all parameters with protected=true in the system.)
- removeSimpleEquations (Performs alias elimination and removes constant variables from the DAE, replacing all occurrences of the old variable reference with the new value (constants) or variable reference (alias elimination).)
- removeUnusedParameter (Strips all parameter not present in the equations from the system.)
- removeUnusedVariables (Strips all variables not present in the equations from the system.)

- removeVerySimpleEquations ([Experimental] Like removeSimpleEquations, but less thorough. Note that this always uses the experimental new alias elimination, --removeSimpleEquations=new, which makes it unstable. In particular, MultiBody systems fail to translate correctly. It can be used for simple (but large) systems of equations.)
- replaceEdgeChange (Replace edge(b) = b and not pre(b) and change(b) = v <> pre(v).)
- residualForm (Transforms simple equations x=y to zero-sum equations 0=y-x.)
- resolveLoops (resolves linear equations in loops)
- simplifyAllExpressions (Does simplifications on all expressions.)
- simplifyIfEquations (Tries to simplify if equations by use of information from evaluated parameters.)
- sortEqnsVars (Heuristic sorting for equations and variables.)
- unitChecking (Does advanced unit checking which consists of two parts: 1. calculation of unspecified unit information for variables; 2. consistency check for all equations based on unit information. Please note: This module is still experimental.)
- wrapFunctionCalls (This module introduces variables for each function call and substitutes all these calls with the newly introduced variables.)

--cheapmatchingAlgorithm

Sets the cheap matching algorithm to use. A cheap matching algorithm gives a jump start matching by heuristics.

Integer (default `3`). Valid options:

- 0 (No cheap matching.)
- 1 (Cheap matching, traverses all equations and match the first free variable.)
- 3 (Random Karp-Sipser: R. M. Karp and M. Sipser. Maximum matching in sparse random graphs.)

--matchingAlgorithm

Sets the matching algorithm to use. See *--help=optmodules* for more info.

String (default PFPlusExt). Valid options:

- BFSB (Breadth First Search based algorithm.)
- DFSB (Depth First Search based algorithm.)
- MC21A (Depth First Search based algorithm with look ahead feature.)
- PF (Depth First Search based algorithm with look ahead feature.)
- PFPlus (Depth First Search based algorithm with look ahead feature and fair row traversal.)
- HK (Combined BFS and DFS algorithm.)
- HKDW (Combined BFS and DFS algorithm.)
- ABMP (Combined BFS and DFS algorithm.)
- PR (Matching algorithm using push relabel mechanism.)

- DFSBExt (Depth First Search based Algorithm external c implementation.)
- BFSBExt (Breadth First Search based Algorithm external c implementation.)
- MC21AExt (Depth First Search based Algorithm with look ahead feature external c implementation.)
- PFExt (Depth First Search based Algorithm with look ahead feature external c implementation.)
- PFPlusExt (Depth First Search based Algorithm with look ahead feature and fair row traversal external c implementation.)
- HKExt (Combined BFS and DFS algorithm external c implementation.)
- HKDWExt (Combined BFS and DFS algorithm external c implementation.)
- ABMPExt (Combined BFS and DFS algorithm external c implementation.)
- PRExt (Matching algorithm using push relabel mechanism external c implementation.)
- BB (BBs try.)

--indexReductionMethod

Sets the index reduction method to use. See *--help=optmodules* for more info.

String (default dynamicStateSelection). Valid options:

- none (Skip index reduction)
- uode (Use the underlying ODE without the constraints.)
- dynamicStateSelection (Simple index reduction method, select (dynamic) dummy states based on analysis of the system.)
- dummyDerivatives (Simple index reduction method, select (static) dummy states based on heuristic.)

--postOptModules

Sets the post optimization modules to use in the back end. See *--help=optmodules* for more info.

String list (default lateInlineFunction,wrapFunctionCalls,inlineArrayEqn,constantLinearSystem,simplifysemiLinear,removeSimpleE
Valid options:

- addScaledVars_states (added var_norm = var/nominal, where var is state)
- addScaledVars_inputs (added var_norm = var/nominal, where var is input)
- addTimeAsState (Experimental feature: this replaces each occurrence of variable time with a new introduced state $time with equation der($time) = 1.0)
- calculateStateSetsJacobians (Generates analytical jacobian for dynamic state selection sets.)
- calculateStrongComponentJacobians (Generates analytical jacobian for torn linear and non-linear strong components. By default linear components and non-linear components with user-defined function calls are skipped. See also debug flags: LSanalyticJacobian, NLSanalyticJacobian and forceNLSanalyticJacobian)
- collapseArrayExpressions (Simplifies {x[1],x[2],x[3]} → x for arrays of whole variable references (simplifies code generation).)

- constantLinearSystem (Evaluates constant linear systems (a*x+b*y=c; d*x+e*y=f; a,b,c,d,e,f are constants) at compile-time.)
- countOperations (Count the mathematical operations of the system.)
- cseBinary (Common Sub-expression Elimination)
- dumpComponentsGraphStr (Dumps the assignment graph used to determine strong components to format suitable for Mathematica)
- dumpDAE (dumps the DAE representation of the current transformation state)
- dumpDAEXML (dumps the DAE as xml representation of the current transformation state)
- evaluateParameters (Evaluates parameters with annotation(Evaluate=true). Use '--evaluateFinalParameters=true' or '--evaluateProtectedParameters=true' to specify additional parameters to be evaluated. Use '--replaceEvaluatedParameters=true' if the evaluated parameters should be replaced in the DAE. To evaluate all parameters in the Frontend use -d=evaluateAllParameters.)
- extendDynamicOptimization (Move loops to constraints.)
- generateSymbolicLinearization (Generates symbolic linearization matrices A,B,C,D for linear model:$\dot{x} = Ax + Bu$)
- generateSymbolicSensitivities (Generates symbolic Sensivities matrix, where der(x) is differentiated w.r.t. param.)
- inlineArrayEqn (This module expands all array equations to scalar equations.)
- inputDerivativesUsed (Checks if derivatives of inputs are need to calculate the model.)
- lateInlineFunction (Perform function inlining for function with annotation LateInline=true.)
- partlintornsystem (partitions linear torn systems.)
- recursiveTearing (inline and repeat tearing)
- reduceDynamicOptimization (Removes equations which are not needed for the calculations of cost and constraints. This module requires -d=reduceDynOpt.)
- relaxSystem (relaxation from gausian elemination)
- removeConstants (Remove all constants in the system.)
- removeEqualRHS (Detects equal function calls of the form a=f(b) and c=f(b) and substitutes them to get speed up.)
- removeSimpleEquations (Performs alias elimination and removes constant variables from the DAE, replacing all occurrences of the old variable reference with the new value (constants) or variable reference (alias elimination).)
- removeUnusedParameter (Strips all parameter not present in the equations from the system to get speed up for compilation of target code.)
- removeUnusedVariables (Strips all variables not present in the equations from the system to get speed up for compilation of target code.)
- reshufflePost (Reshuffles algebraic loops.)

- simplifyAllExpressions (Does simplifications on all expressions.)
- simplifyComplexFunction (Some simplifications on complex functions (complex refers to the internal data structure))
- simplifyConstraints (Rewrites nonlinear constraints into box constraints if possible. This module requires +gDynOpt.)
- simplifyLoops (Simplifies algebraic loops. This modules requires +simplifyLoops.)
- simplifyTimeIndepFuncCalls (Simplifies time independent built in function calls like pre(param) -> param, der(param) -> 0.0, change(param) -> false, edge(param) -> false.)
- simplifysemiLinear (Simplifies calls to semiLinear.)
- solveLinearSystem (solve linear system with newton step)
- solveSimpleEquations (Solves simple equations)
- symSolver (Rewrites the ode system for implicit Euler method. This module requires +symSolver.)
- symbolicJacobian (Detects the sparse pattern of the ODE system and calculates also the symbolic Jacobian if flag '--generateSymbolicJacobian' is enabled.)
- tearingSystem (For method selection use flag tearingMethod.)
- wrapFunctionCalls (This module introduces variables for each function call and substitutes all these calls with the newly introduced variables.)

--simCodeTarget

Sets the target language for the code generation.

String (default C). Valid options:

- None
- Adevs
- C
- Cpp
- CSharp
- ExperimentalEmbeddedC
- Java
- JavaScript
- omsic
- sfmi
- XML
- MidC

--orderConnections

Orders connect equations alphabetically if set.

Boolean (default `true`).

-t, --typeinfo

Prints out extra type information if set.

Boolean (default `false`).

-a, --keepArrays

Sets whether to split arrays or not.

Boolean (default `false`).

-m, --modelicaOutput

Enables valid modelica output for flat modelica.

Boolean (default `false`).

-q, --silent

Turns on silent mode.

Boolean (default `false`).

-c, --corbaSessionName

Sets the name of the corba session if -d=interactiveCorba or --interactive=corba is used.

String (default *empty*).

-n, --numProcs

Sets the number of processors to use (0=default=auto).

Integer (default `0`).

-l, --latency

Sets the latency for parallel execution.

Integer (default `0`).

-b, --bandwidth

Sets the bandwidth for parallel execution.

Integer (default `0`).

-i, --instClass

Instantiate the class given by the fully qualified path.

String (default *empty*).

-v, --vectorizationLimit

Sets the vectorization limit, arrays and matrices larger than this will not be vectorized.

Integer (default `0`).

-s, --simulationCg

Turns on simulation code generation.

Boolean (default `false`).

--evalAnnotationParams

Sets whether to evaluate parameters in annotations or not.

Boolean (default `false`).

--generateLabeledSimCode

Turns on labeled SimCode generation for reduction algorithms.

Boolean (default `false`).

--reduceTerms

Turns on reducing terms for reduction algorithms.

Boolean (default `false`).

--reductionMethod

Sets the reduction method to be used.

String (default deletion). Valid options:

- deletion
- substitution
- linearization

--demoMode

Disable Warning/Error Massages.

Boolean (default `false`).

--locale

Override the locale from the environment.

String (default *empty*).

-o, --defaultOCLDevice

Sets the default OpenCL device to be used for parallel execution.

Integer (default `0`).

--maxTraversals

Maximal traversals to find simple equations in the acausal system.

Integer (default `2`).

--dumpTarget

Redirect the dump to file. If the file ends with .html HTML code is generated.

String (default *empty*).

--delayBreakLoop

Enables (very) experimental code to break algebraic loops using the delay() operator. Probably messes with initialization.

Boolean (default `true`).

--tearingMethod

Sets the tearing method to use. Select no tearing or choose tearing method.

String (default cellier). Valid options:

- noTearing (Skip tearing.)
- minimalTearing (Minimal tearing method based on a brute force approuch.)
- omcTearing (Tearing method developed by TU Dresden: Frenkel, Schubert.)
- cellier (Tearing based on Celliers method, revised by FH Bielefeld: Täuber, Patrick)

--tearingHeuristic

Sets the tearing heuristic to use for Cellier-tearing.

String (default MC3). Valid options:

- MC1 (Original cellier with consideration of impossible assignments and discrete Vars.)
- MC2 (Modified cellier, drop first step.)
- MC11 (Modified MC1, new last step 'count impossible assignments'.)
- MC21 (Modified MC2, new last step 'count impossible assignments'.)
- MC12 (Modified MC1, step 'count impossible assignments' before last step.)
- MC22 (Modified MC2, step 'count impossible assignments' before last step.)
- MC13 (Modified MC1, build sum of impossible assignment and causalizable equations, choose var with biggest sum.)
- MC23 (Modified MC2, build sum of impossible assignment and causalizable equations, choose var with biggest sum.)
- MC231 (Modified MC23, Two rounds, choose better potentials-set.)
- MC3 (Modified cellier, build sum of impossible assignment and causalizable equations for all vars, choose var with biggest sum.)
- MC4 (Modified cellier, use all heuristics, choose var that occurs most in potential sets)

--disableLinearTearing

Disables the tearing of linear systems. That might improve the performance of large linear systems(N>1000) in combination with a sparse solver (e.g. umfpack) at runtime (usage with: -ls umfpack). Deprecated flag: Use --maxSizeLinearTearing=0 instead.

Boolean (default `false`).

--scalarizeMinMax

Scalarizes the builtin min/max reduction operators if true.

Boolean (default `false`).

--scalarizeBindings

Always scalarizes bindings if set.

Boolean (default `false`).

--corbaObjectReferenceFilePath

Sets the path for corba object reference file if -d=interactiveCorba is used.

String (default *empty*).

--hpcomScheduler

Sets the scheduler for task graph scheduling (list | listr | level | levelfix | ext | metis | mcp | taskdep | tds | bls | rand | none). Default: level.

String (default level).

--hpcomCode

Sets the code-type produced by hpcom (openmp | pthreads | pthreads_spin | tbb | mpi). Default: openmp.

String (default openmp).

--rewriteRulesFile

Activates user given rewrite rules for Absyn expressions. The rules are read from the given file and are of the form rewrite(fromExp, toExp);

String (default *empty*).

--replaceHomotopy

Replaces homotopy(actual, simplified) with the actual expression or the simplified expression. Good for debugging models which use homotopy. The default is to not replace homotopy.

String (default none). Valid options:

- none (Default, do not replace homotopy.)
- actual (Replace homotopy(actual, simplified) with actual.)
- simplified (Replace homotopy(actual, simplified) with simplified.)

--generateSymbolicJacobian

Generates symbolic Jacobian matrix, where der(x) is differentiated w.r.t. x. This matrix can be used by dassl or ida solver with simulation flag '-jacobian'.

Boolean (default `false`).

--generateSymbolicLinearization

Generates symbolic linearization matrices A,B,C,D for linear model: $\dot{x} = Ax + Bu\ y = Cx + Du$

Boolean (default `false`).

--intEnumConversion

Allow Integer to enumeration conversion.

Boolean (default `false`).

--profiling

Sets the profiling level to use. Profiled equations and functions record execution time and count for each time step taken by the integrator.

String (default none). Valid options:

- none (Generate code without profiling)
- blocks (Generate code for profiling function calls as well as linear and non-linear systems of equations)
- blocks+html (Like blocks, but also run xsltproc and gnuplot to generate an html report)
- all (Generate code for profiling of all functions and equations)
- all_perf (Generate code for profiling of all functions and equations with additional performance data using the papi-interface (cpp-runtime))
- all_stat (Generate code for profiling of all functions and equations with additional statistics (cpp-runtime))

--reshuffle

sets tolerance of reshuffling algorithm: 1: conservative, 2: more tolerant, 3 resolve all

Integer (default `1`).

--gDynOpt

Generate dynamic optimization problem based on annotation approach.

Boolean (default `false`).

--maxSizeSolveLinearSystem

Max size for solveLinearSystem.

Integer (default `0`).

--cppFlags

Sets extra flags for compilation with the C++ compiler (e.g. +cppFlags=-O3,-Wall)

String list (default).

--removeSimpleEquations

Specifies method that removes simple equations.

String (default default). Valid options:

- none (Disables module)
- default (Performs alias elimination and removes constant variables. Default case uses in preOpt phase the fastAcausal and in postOpt phase the causal implementation.)
- causal (Performs alias elimination and removes constant variables. Causal implementation.)
- fastAcausal (Performs alias elimination and removes constant variables. fastImplementation fastAcausal.)
- allAcausal (Performs alias elimination and removes constant variables. Implementation allAcausal.)
- new (New implementation (experimental))

--dynamicTearing

Activates dynamic tearing (TearingSet can be changed automatically during runtime, strict set vs. casual set.)

String (default false). Valid options:

- false (No dynamic tearing.)
- true (Dynamic tearing for linear and nonlinear systems.)
- linear (Dynamic tearing only for linear systems.)
- nonlinear (Dynamic tearing only for nonlinear systems.)

--symSolver

Activates symbolic implicit solver (original system is not changed).

String (default none). Valid options:

- none
- impEuler
- expEuler

--loop2con

Specifies method that transform loops in constraints. hint: using initial guess from file!

String (default none). Valid options:

- none (Disables module)
- lin (linear loops --> constraints)
- noLin (no linear loops --> constraints)
- all (loops --> constraints)

--forceTearing

Use tearing set even if it is not smaller than the original component.

Boolean (default `false`).

--simplifyLoops

Simplify algebraic loops.

Integer (default `0`). Valid options:

- 0 (do nothing)
- 1 (special modification of residual expressions)
- 2 (special modification of residual expressions with helper variables)

--recursiveTearing

Inline and repeat tearing.

Integer (default `0`). Valid options:

- 0 (do nothing)
- 1 (linear tearing set of size 1)
- 2 (linear tearing)

--flowThreshold

Sets the minium threshold for stream flow rates

Real (default `1e-07`).

--matrixFormat

Sets the matrix format type in cpp runtime which should be used (dense | sparse). Default: dense.

String (default dense).

--partlintorn

Sets the limit for partitionin of linear torn systems.

Integer (default `0`).

--initOptModules

Sets the initialization optimization modules to use in the back end. See *--help=optmodules* for more info.

String list (default simplifyComplexFunction,tearingSystem,solveSimpleEquations,calculateStrongComponentJacobians,simplifyA
Valid options:

- calculateStrongComponentJacobians (Generates analytical jacobian for torn linear and non-linear strong components. By default linear components and non-linear components with user-defined function calls are skipped. See also debug flags: LSanalyticJacobian, NLSanalyticJacobian and forceNLSanalyticJacobian)
- collapseArrayExpressions (Simplifies {x[1],x[2],x[3]} → x for arrays of whole variable references (simplifies code generation).)

- constantLinearSystem (Evaluates constant linear systems (a*x+b*y=c; d*x+e*y=f; a,b,c,d,e,f are constants) at compile-time.)
- extendDynamicOptimization (Move loops to constraints.)
- generateHomotopyComponents (Finds the parts of the DAE that have to be handled by the homotopy solver and creates a strong component out of it.)
- inlineHomotopy (Experimental: Inlines the homotopy expression to allow symbolic simplifications.)
- inputDerivativesUsed (Checks if derivatives of inputs are need to calculate the model.)
- recursiveTearing (inline and repeat tearing)
- reduceDynamicOptimization (Removes equations which are not needed for the calculations of cost and constraints. This module requires -d=reduceDynOpt.)
- replaceHomotopyWithSimplified (Replaces the homotopy expression homotopy(actual, simplified) with the simplified part.)
- simplifyAllExpressions (Does simplifications on all expressions.)
- simplifyComplexFunction (Some simplifications on complex functions (complex refers to the internal data structure))
- simplifyConstraints (Rewrites nonlinear constraints into box constraints if possible. This module requires +gDynOpt.)
- simplifyLoops (Simplifies algebraic loops. This modules requires +simplifyLoops.)
- solveSimpleEquations (Solves simple equations)
- tearingSystem (For method selection use flag tearingMethod.)
- wrapFunctionCalls (This module introduces variables for each function call and substitutes all these calls with the newly introduced variables.)

--maxMixedDeterminedIndex

Sets the maximum mixed-determined index that is handled by the initialization.

Integer (default `10`).

--useLocalDirection

Keeps the input/output prefix for all variables in the flat model, not only top-level ones.

Boolean (default `false`).

--defaultOptModulesOrdering

If this is activated, then the specified pre-/post-/init-optimization modules will be rearranged to the recommended ordering.

Boolean (default `true`).

--preOptModules+

Enables additional pre-optimization modules, e.g. --preOptModules+=module1,module2 would additionally enable module1 and module2. See *--help=optmodules* for more info.

String list (default *empty*).

--preOptModules-

Disables a list of pre-optimization modules, e.g. --preOptModules-=module1,module2 would disable module1 and module2. See *--help=optmodules* for more info.

String list (default *empty*).

--postOptModules+

Enables additional post-optimization modules, e.g. --postOptModules+=module1,module2 would additionally enable module1 and module2. See *--help=optmodules* for more info.

String list (default *empty*).

--postOptModules-

Disables a list of post-optimization modules, e.g. --postOptModules-=module1,module2 would disable module1 and module2. See *--help=optmodules* for more info.

String list (default *empty*).

--initOptModules+

Enables additional init-optimization modules, e.g. --initOptModules+=module1,module2 would additionally enable module1 and module2. See *--help=optmodules* for more info.

String list (default *empty*).

--initOptModules-

Disables a list of init-optimization modules, e.g. --initOptModules-=module1,module2 would disable module1 and module2. See *--help=optmodules* for more info.

String list (default *empty*).

--instCacheSize

Sets the size of the internal hash table used for instantiation caching.

Integer (default `25343`).

--maxSizeLinearTearing

Sets the maximum system size for tearing of linear systems (default 200).

Integer (default `200`).

--maxSizeNonlinearTearing

Sets the maximum system size for tearing of nonlinear systems (default 10000).

Integer (default `10000`).

--noTearingForComponent

Deactivates tearing for the specified components. Use '-d=tearingdump' to find out the relevant indexes.

Unknown default valueFlags.FlagData.INT_LIST_FLAG(data = {NIL})

--daeMode

Generates code to simulate models in DAE mode. The whole system is passed directly to the DAE solver SUNDIALS/IDA and no algebraic solver is involved in the simulation process.

Boolean (default `false`).

--inlineMethod

Sets the inline method to use. replace : This method inlines by replacing in place all expressions. Might lead to very long expression. append : This method inlines by adding additional variables to the whole system. Might lead to much bigger system.

String (default replace). Valid options:

- replace
- append

--setTearingVars

Sets the tearing variables by its strong component indexes. Use '-d=tearingdump' to find out the relevant indexes. Use following format: '--setTearingVars=(sci,n,t1,...,tn)*', with sci = strong component index, n = number of tearing variables, t1,...tn = tearing variables. E.g.: '--setTearingVars=4,2,3,5' would select variables 3 and 5 in strong component 4.

Unknown default valueFlags.FlagData.INT_LIST_FLAG(data = {NIL})

--setResidualEqns

Sets the residual equations by its strong component indexes. Use '-d=tearingdump' to find out the relevant indexes for the collective equations. Use following format: '--setResidualEqns=(sci,n,r1,...,rn)*', with sci = strong component index, n = number of residual equations, r1,...rn = residual equations. E.g.: '--setResidualEqns=4,2,3,5' would select equations 3 and 5 in strong component 4. Only works in combination with 'setTearingVars'.

Unknown default valueFlags.FlagData.INT_LIST_FLAG(data = {NIL})

--ignoreCommandLineOptionsAnnotation

Ignores the command line options specified as annotation in the class.

Boolean (default `false`).

--calculateSensitivities

Generates sensitivities variables and matrixes.

Boolean (default `false`).

-r, --alarm

Sets the number seconds until omc timeouts and exits. Used by the testing framework to terminate infinite running processes.

Integer (default `0`).

--totalTearing

Activates total tearing (determination of all possible tearing sets) for the specified components. Use '-d=tearingdump' to find out the relevant indexes.

Unknown default valueFlags.FlagData.INT_LIST_FLAG(data = {NIL})

--ignoreSimulationFlagsAnnotation

Ignores the simulation flags specified as annotation in the class.

Boolean (default `false`).

--dynamicTearingForInitialization

Enable Dynamic Tearing also for the initialization system.

Boolean (default `false`).

--preferTVarsWithStartValue

Prefer tearing variables with start value for initialization.

Boolean (default `true`).

--equationsPerFile

Generate code for at most this many equations per C-file (partially implemented in the compiler).

Integer (default `2000`).

--evaluateFinalParameters

Evaluates all the final parameters in addition to parameters with annotation(Evaluate=true).

Boolean (default `false`).

--evaluateProtectedParameters

Evaluates all the protected parameters in addition to parameters with annotation(Evaluate=true).

Boolean (default `false`).

--replaceEvaluatedParameters

Replaces all the evaluated parameters in the DAE.

Boolean (default `true`).

--condenseArrays

Sets whether array expressions containing function calls are condensed or not.

Boolean (default `true`).

--wfcAdvanced

wrapFunctionCalls ignores more then default cases, e.g. exp, sin, cos, log, (experimental flag)

Boolean (default `false`).

--tearingStrictness

Sets the strictness of the tearing method regarding the solvability restrictions.

String (default strict). Valid options:

- casual (Loose tearing rules using ExpressionSolve to determine the solvability instead of considering the partial derivative. Allows to solve for everything that is analytically possible. This could lead to singularities during simulation.)
- strict (Robust tearing rules by consideration of the partial derivative. Allows to divide by parameters that are not equal to or close to zero.)
- veryStrict (Very strict tearing rules that do not allow to divide by any parameter. Use this if you aim at overriding parameters after compilation with values equal to or close to zero.)

--interactive

Sets the interactive mode for omc.

String (default none). Valid options:

- none (do nothing)
- corba (Starts omc as a server listening on the socket interface.)
- tcp (Starts omc as a server listening on the Corba interface.)
- zmq (Starts omc as a ZeroMQ server listening on the socket interface.)

-z, --zeroMQFileSuffix

Sets the file suffix for zeroMQ port file if --interactive=zmq is used.

String (default *empty*).

--homotopyApproach

Sets the homotopy approach.

String (default equidistantGlobal). Valid options:

- equidistantLocal (Local homotopy approach with equidistant lambda steps. The homotopy parameter only effects the local strongly connected component.)
- adaptiveLocal (Local homotopy approach with adaptive lambda steps. The homotopy parameter only effects the local strongly connected component.)
- equidistantGlobal (Default, global homotopy approach with equidistant lambda steps. The homotopy parameter effects the entire initialization system.)
- adaptiveGlobal (Global homotopy approach with adaptive lambda steps. The homotopy parameter effects the entire initialization system.)

--ignoreReplaceable

Sets whether to ignore replaceability or not when redeclaring.

Boolean (default `false`).

--postOptModulesDAE

Sets the optimization modules for the DAEmode in the back end. See *--help=optmodules* for more info.

String list (default lateInlineFunction,wrapFunctionCalls,simplifysemiLinear,simplifyComplexFunction,removeConstants,simplifyT

--evalLoopLimit

The loop iteration limit used when evaluating constant function calls.

Integer (default `100000`).

--evalRecursionLimit

The recursion limit used when evaluating constant function calls.

Integer (default `256`).

--singleInstanceAglSolver

Sets to instantiate only one algebraic loop solver all algebraic loops

Boolean (default `false`).

--showStructuralAnnotations

Show annotations affecting the solution process in the flattened code.

Boolean (default `false`).

--initialStateSelection

Activates the state selection inside initialization to avoid singularities.

Boolean (default `false`).

--strict

Enables stricter enforcement of Modelica language rules.

Boolean (default `false`).

--linearizationDumpLanguage

Sets the target language for the produced code of linearization. Only works with '--generateSymbolicLinearization' and 'linearize(modelName)'.

String (default modelica). Valid options:

- modelica
- matlab
- julia
- python

--convertAnalyticalSingularities

Allows the compiler to try to convert analytical to structural singularities.

Boolean (default `false`).

24.2 Debug flags

The debug flag takes a comma-separated list of flags which are used by the compiler for debugging or experimental purposes. Flags prefixed with "-" or "no" will be disabled. The available flags are (+ are enabled by default, - are disabled):

Cache **(default: on)** Turns off the instantiation cache.

LSanalyticJacobian **(default: off)** Enables analytical jacobian for linear strong components. Defaults to false

NLSanalyticJacobian **(default: on)** Enables analytical jacobian for non-linear strong components without user-defined function calls, for that see forceNLSanalyticJacobian

acceptTooManyFields **(default: off)** Accepts passing records with more fields than expected to a function. This is not allowed, but is used in Fluid.Dissipation. See https://trac.modelica.org/Modelica/ticket/1245 for details.

addDerAliases **(default: off)** Adds for every der-call an alias equation e.g. dx = der(x). It's a work-a-round flag, which helps in some cases to simulate the models e.g. Modelica.Fluid.Examples.HeatExchanger.HeatExchangerSimulation. Deprecated flag: Use --preOptModules+=introduceDerAlias instead.

addScaledVars **(default: off)** Adds an alias equation var_nrom = var/nominal where var is state Deprecated flag: Use --postOptModules+=addScaledVars_states instead.

addScaledVarsInput **(default: off)** Adds an alias equation var_nrom = var/nominal where var is input Deprecated flag: Use --postOptModules+=addScaledVars_inputs instead.

aliasConflicts **(default: off)** Dumps alias sets with different start or nominal values.

backendKeepEnv **(default: on)** When enabled, the environment is kept when entering the backend, which enables CevalFunction (function interpretation) to work. This module not essential for the backend to function in most cases, but can improve simulation performance by evaluating functions. The drawback to keeping the environment graph in memory is that it is huge (~80% of the total memory in use when returning the frontend DAE).

backendReduceDAE **(default: off)** Prints all Reduce DAE debug information.

backenddaeinfo **(default: off)** Enables dumping of back-end information about system (Number of equations before back-end,...).

bltdump **(default: off)** Dumps information from index reduction.

bltmatrixdump **(default: off)** Dumps the blt matrix in html file. IE seems to be very good in displaying large matrices.

buildExternalLibs **(default: on)** Use the autotools project in the Resources folder of the library to build missing external libraries.

ceval **(default: off)** Prints extra information from Ceval.

cgraph **(default: off)** Prints out connection graph information.

cgraphGraphVizFile **(default: off)** Generates a graphviz file of the connection graph.

cgraphGraphVizShow **(default: off)** Displays the connection graph with the GraphViz lefty tool.

checkASUB **(default: off)** Prints out a warning if an ASUB is created from a CREF expression.

checkBackendDae **(default: off)** Do some simple analyses on the datastructure from the frontend to check if it is consistent.

checkDAECrefType **(default: off)** Enables extra type checking for cref expressions.

checkSimplify **(default: off)** Enables checks for expression simplification and prints a notification whenever an undesirable transformation has been performed.

constjac **(default: off)** solves linear systems with constant Jacobian and variable b-Vector symbolically

convertAnalyticalDump **(default: off)** Dumps the conversion process of analytical to structural singularities.

countOperations **(default: off)** Count operations.

daedumpgraphv **(default: off)** Dumps the DAE in graphviz format.

debugAlgebraicLoopsJacobian **(default: off)** Dumps debug output while creating symbolic jacobians for non-linear systems.

debugAlias **(default: off)** Dumps some information about the process of removeSimpleEquations.

debugDAEmode **(default: off)** Dump debug output for the DAEmode.

debugDifferentiation **(default: off)** Dumps debug output for the differentiation process.

debugDifferentiationVerbose **(default: off)** Dumps verbose debug output for the differentiation process.

disableColoring **(default: off)** Disables coloring algorithm while spasity detection.

disableComSubExp **(default: off)** Deactivates module 'comSubExp' Deprecated flag: Use --preOptModules-=comSubExp instead.

disableDirectionalDerivatives **(default: on)** For FMI 2.0 only dependecy analysis will be perform.

disableFMIDependency **(default: off)** Disables the dependency analysis and generation for FMI 2.0.

disableJacsforSCC **(default: off)** Disables calculation of jacobians to detect if a SCC is linear or non-linear. By disabling all SCC will handled like non-linear.

disablePartitioning **(default: off)** Deactivates partitioning of entire equation system. Deprecated flag: Use --preOptModules-=clockPartitioning instead.

disableRecordConstructorOutput **(default: off)** Disables output of record constructors in the flat code.

disableSimplifyComplexFunction **(default: off)** disable simplifyComplexFunction Deprecated flag: Use --postOptModules-=simplifyComplexFunction/--initOptModules-=simplifyComplexFunction instead.

disableSingleFlowEq **(default: off)** Disables the generation of single flow equations.

disableStartCalc **(default: off)** Deactivates the pre-calculation of start values during compile-time.

disableWindowsPathCheckWarning **(default: off)** Disables warnings on Windows if OPENMODELICAHOME/MinGW is missing.

discreteinfo **(default: off)** Enables dumping of discrete variables. Extends -d=backenddaeinfo.

dummyselect **(default: off)** Dumps information from dummy state selection heuristic.

dump **(default: off)** Dumps the absyn representation of a program.

dumpBackendInline **(default: off)** Dumps debug output while inline function.

dumpBackendInlineVerbose **(default: off)** Dumps debug output while inline function.

dumpCSE **(default: off)** Additional output for CSE module.

dumpCSE_verbose **(default: off)** Additional output for CSE module.

dumpConstrepl **(default: off)** Dump the found replacements for constants.

dumpEArepl **(default: off)** Dump the found replacements for evaluate annotations (evaluate=true) parameters.

dumpEncapsulateConditions **(default: off)** Dumps the results of the preOptModule encapsulateWhenConditions.

dumpEqInUC **(default: off)** Dumps all equations handled by the unit checker.

dumpEqUCStruct **(default: off)** Dumps all the equations handled by the unit checker as tree-structure.

dumpExcludedSymJacExps **(default: off)** This flags dumps all expression that are excluded from differentiation of a symbolic Jacobian.

dumpFPrepl **(default: off)** Dump the found replacements for final parameters.

dumpFunctions **(default: off)** Add functions to backend dumps.

dumpHomotopy **(default: off)** Dumps the results of the postOptModule optimizeHomotopyCalls.

dumpInlineSolver **(default: off)** Dumps the inline solver equation system.

dumpJL **(default: off)** Dumps the absyn representation of a program as a Julia representation

dumpLoops **(default: off)** Dumps loop equation.

dumpPPrepl **(default: off)** Dump the found replacements for protected parameters.

dumpParamrepl **(default: off)** Dump the found replacements for remove parameters.

dumpRecursiveTearing **(default: off)** Dump between steps of recursiveTearing

dumpSCCGraphML **(default: off)** Dumps graphml files with the strongly connected components.

dumpSimCode **(default: off)** Dumps the simCode model used for code generation.

dumpSimplifyLoops **(default: off)** Dump between steps of simplifyLoops

dumpSortEqnsAndVars **(default: off)** Dumps debug output for the modules sortEqnsVars.

dumpSparsePattern **(default: off)** Dumps sparse pattern with coloring used for simulation.

dumpSparsePatternVerbose **(default: off)** Dumps in verbose mode sparse pattern with coloring used for simulation.

dumpSynchronous **(default: off)** Dumps information of the clock partitioning.

dumpTransformedModelica **(default: off)** Dumps the back-end DAE to a Modelica-like model after all symbolic transformations are applied.

dumpUnits **(default: off)** Dumps all the calculated units.

dumpdaelow **(default: off)** Dumps the equation system at the beginning of the back end.

dumpdgesv **(default: off)** Enables dumping of the information whether DGESV is used to solve linear systems.

dumpeqninorder **(default: off)** Enables dumping of the equations in the order they are calculated.

dumpindxdae **(default: off)** Dumps the equation system after index reduction and optimization.

dumpinitialsystem **(default: off)** Dumps the initial equation system.

dumprepl **(default: off)** Dump the found replacements for simple equation removal.

dynload **(default: off)** Display debug information about dynamic loading of compiled functions.

evalConstFuncs **(default: on)** Evaluates functions complete and partially and checks for constant output. Deprecated flag: Use --preOptModules+=evalFunc instead.

evalFuncDump **(default: off)** dumps debug information about the function evaluation

evalOutputOnly **(default: off)** Generates equations to calculate outputs only.

evalParameterDump **(default: off)** Dumps information for evaluating parameters.

evalfunc **(default: on)** Turns on/off symbolic function evaluation.

evaluateAllParameters **(default: off)** Evaluates all parameters if set.

events **(default: on)** Turns on/off events handling.

execHash **(default: off)** Measures the time it takes to hash all simcode variables before code generation.

execstat **(default: off)** Prints out execution statistics for the compiler.

execstatGCcollect **(default: off)** When running execstat, also perform an extra full garbage collection.

experimentalReductions **(default: off)** Turns on custom reduction functions (OpenModelica extension).

failtrace **(default: off)** Sets whether to print a failtrace or not.

fmuExperimental **(default: off)** Include an extra function in the FMU fmi2GetSpecificDerivatives.

forceNLSanalyticJacobian **(default: off)** Forces calculation analytical jacobian also for non-linear strong components with user-defined functions.

frontEndUnitCheck **(default: off)** Checks the consistency of units in equation.

gcProfiling **(default: off)** Prints garbage collection stats to standard output.

gen **(default: off)** Turns on/off dynamic loading of functions that are compiled during translation. Only enable this if external functions are needed to calculate structural parameters or constants.

gendebugsymbols **(default: off)** Generate code with debugging symbols.

generateCodeCheat **(default: off)** Used to generate code for the bootstrapped compiler.

graphInst **(default: off)** Do graph based instantiation.

graphInstGenGraph **(default: off)** Dumps a graph of the program. Use with -d=graphInst

graphInstRunDep **(default: off)** Run scode dependency analysis. Use with -d=graphInst

graphInstShowGraph **(default: off)** Display a graph of the program interactively. Use with -d=graphInst

graphml **(default: off)** Dumps .graphml files for the bipartite graph after Index Reduction and a task graph for the SCCs. Can be displayed with yEd.

graphviz **(default: off)** Dumps the absyn representation of a program in graphviz format.

graphvizDump **(default: off)** Activates additional graphviz dumps (as .dot files). It can be used in addition to one of the following flags: {dumpdaelow|dumpinitialsystems|dumpindxdae}.

hardcodedStartValues **(default: off)** Embed the start values of variables and parameters into the c++ code and do not read it from xml file.

hpcom **(default: off)** Enables parallel calculation based on task-graphs.

hpcomDump **(default: off)** Dumps additional information on the parallel execution with hpcom.

hpcomMemoryOpt **(default: off)** Optimize the memory structure regarding the selected scheduler

ignoreCycles **(default: off)** Ignores cycles between constant/parameter components.

implOde **(default: off)** activates implicit codegen

infoXmlOperations **(default: off)** Enables output of the operations in the _info.xml file when translating models.

initialization **(default: off)** Shows additional information from the initialization process.

inlineFunctions **(default: on)** Controls if function inlining should be performed.

inlineSolver **(default: off)** Generates code for inline solver.

instance **(default: off)** Prints extra failtrace from InstanceHierarchy.

interactive **(default: off)** Starts omc as a server listening on the socket interface.

interactiveCorba **(default: off)** Starts omc as a server listening on the Corba interface.

interactivedump **(default: off)** Prints out debug information for the interactive server.

iterationVars **(default: off)** Shows a list of all iteration variables.

listAppendWrongOrder **(default: on)** Print notifications about bad usage of listAppend.

lookup **(default: off)** Print extra failtrace from lookup.

mergeAlgSections **(default: off)** Disables coloring algorithm while sparsity detection.

metaModelicaRecordAllocWords **(default: off)** Instrument the source code to record memory allocations (requires run-time and generated files compiled with -DOMC_RECORD_ALLOC_WORDS).

multirate **(default: off)** The solver can switch partitions in the system.

newInst **(default: off)** Enables experimental new instantiation phase.

nfAPI **(default: off)** Enables experimental new instantiation use in the OMC API.

nfAPIDynamicSelect **(default: off)** Show DynamicSelect(static, dynamic) in annotations. Default to false and will select the first (static) expression

nfAPINoise **(default: off)** Enables error display for the experimental new instantiation use in the OMC API.

nfEvalConstArgFuncs **(default: on)** Evaluate all functions with constant arguments in the new frontend.

nfExpandFuncArgs **(default: off)** Expand all function arguments in the new frontend.

nfExpandOperations **(default: on)** Expand all unary/binary operations to scalar expressions in the new frontend.

nfScalarize **(default: on)** Run scalarization in NF, default true.

oldFrontEndUnitCheck **(default: off)** Checks the consistency of units in equation (for the old front-end).

onRelaxation **(default: off)** Perform O(n) relaxation. Deprecated flag: Use --postOptModules+=relaxSystem instead.

optdaedump **(default: off)** Dumps information from the optimization modules.

parallelCodegen **(default: on)** Enables code generation in parallel (disable this if compiling a model causes you to run out of RAM).

paramdlowdump **(default: off)** Enables dumping of the parameters in the order they are calculated.

parmodauto **(default: off)** Experimental: Enable parallelization of independent systems of equations in the translated model.

partitionInitialization **(default: on)** This flag controls if partitioning is applied to the initialization system.

patternmAllInfo **(default: off)** Adds notifications of all pattern-matching optimizations that are performed.

patternmDeadCodeElimination **(default: on)** Performs dead code elimination in match-expressions.

patternmMoveLastExp **(default: on)** Optimization that moves the last assignment(s) into the result of a match-expression. For example: equation c = fn(b); then c; => then fn(b);

patternmSkipFilterUnusedBindings (default: off)

printStructuralParameters **(default: off)** Prints the structural parameters identified by the front-end

pthreads **(default: off)** Experimental: Unused parallelization.

reduceDynOpt **(default: off)** remove eqs which not need for the calculations of cost and constraints Deprecated flag: Use --postOptModules+=reduceDynamicOptimization instead.

relidx **(default: off)** Prints out debug information about relations, that are used as zero crossings.

relocatableFunctions **(default: off)** Generates relocatable code: all functions become function pointers and can be replaced at run-time.

reportSerializedSize **(default: off)** Reports serialized sizes of various data structures used in the compiler.

reshufflePost **(default: off)** Reshuffles the systems of equations.

resolveLoopsDump **(default: off)** Debug Output for ResolveLoops Module.

rml **(default: off)** Converts Modelica-style arrays to lists.

runtimeStaticLinking **(default: off)** Use the static simulation runtime libraries (C++ simulation runtime).

scodeDep **(default: on)** Does scode dependency analysis prior to instantiation. Defaults to true.

semiLinear **(default: off)** Enables dumping of the optimization information when optimizing calls to semiLinear.

shortOutput **(default: off)** Enables short output of the simulate() command. Useful for tools like OMNotebook.

showDaeGeneration **(default: off)** Show the dae variable declarations as they happen.

showEquationSource **(default: off)** Display the element source information in the dumped DAE for easier debugging.

showExpandableInfo **(default: off)** Show information about expandable connector handling.

showInstCacheInfo **(default: off)** Prints information about instantiation cache hits and additions. Defaults to false.

showStartOrigin **(default: off)** Enables dumping of the DAE startOrigin attribute of the variables.

showStatement **(default: off)** Shows the statement that is currently being evaluated when evaluating a script.

skipInputOutputSyntacticSugar **(default: off)** Used when bootstrapping to preserve the input output parsing of the code output by the list command.

stateselection **(default: off)** Enables dumping of selected states. Extends -d=backenddaeinfo.

static **(default: off)** Enables extra debug output from the static elaboration.

stripPrefix **(default: on)** Strips the environment prefix from path/crefs. Defaults to true.

susanDebug **(default: off)** Makes Susan generate code using try/else to better debug which function broke the expected match semantics.

symJacConstantSplit **(default: off)** Generates all symbolic Jacobians with splitted constant parts.

symjacdump **(default: off)** Dumps information about symbolic Jacobians. Can be used only with postOptModules: generateSymbolicJacobian, generateSymbolicLinearization.

symjacdumpeqn **(default: off)** Dump for debug purpose of symbolic Jacobians. (deactivated now).

symjacdumpverbose **(default: off)** Dumps information in verbose mode about symbolic Jacobians. Can be used only with postOptModules: generateSymbolicJacobian, generateSymbolicLinearization.

symjacwarnings **(default: off)** Prints warnings regarding symoblic jacbians.

tail **(default: off)** Prints out a notification if tail recursion optimization has been applied.

tearingdump **(default: off)** Dumps tearing information.

tearingdumpV **(default: off)** Dumps verbose tearing information.

totaltearingdump **(default: off)** Dumps total tearing information.

totaltearingdumpV **(default: off)** Dumps verbose total tearing information.

tplPerfTimes **(default: off)** Enables output of template performance data for rendering text to file.

transformsbeforedump **(default: off)** Applies transformations required for code generation before dumping flat code.

types **(default: off)** Prints extra failtrace from Types.

uncertainties **(default: off)** Enables dumping of status when calling modelEquationsUC.

updmod **(default: off)** Prints information about modification updates.

useMPI **(default: off)** Add MPI init and finalize to main method (CPPruntime).

vectorize **(default: off)** Activates vectorization in the backend.

visxml **(default: off)** Outputs a xml-file that contains information for visualization.

warnMinMax **(default: on)** Makes a warning assert from min/max variable attributes instead of error.

warnNoNominal **(default: off)** Prints the iteration variables in the initialization and simulation DAE, which do not have a nominal value.

writeToBuffer **(default: off)** Enables writing simulation results to buffer.

24.3 Flags for Optimization Modules

Flags that determine which symbolic methods are used to produce the causalized equation system.

The *--preOptModules* flag sets the optimization modules which are used before the matching and index reduction in the back end. These modules are specified as a comma-separated list.

The *--matchingAlgorithm* sets the method that is used for the matching algorithm, after the pre optimization modules.

The *--indexReductionMethod* sets the method that is used for the index reduction, after the pre optimization modules.

The *--initOptModules* then sets the optimization modules which are used after the index reduction to optimize the system for initialization, specified as a comma-separated list.

The *--postOptModules* then sets the optimization modules which are used after the index reduction to optimize the system for simulation, specified as a comma-separated list.

第25章　Small Overview of Simulation Flags

This chapter contains a *short overview of simulation flags* as well as additional details of the *numerical integration methods*.

25.1 OpenModelica (C-runtime) Simulation Flags

The simulation executable takes the following flags:

-abortSlowSimulation Aborts if the simulation chatters.

***-alarm=value* *or* -alarm value** Aborts after the given number of seconds (default=0 disables the alarm).

***-clock=value* *or* -clock value** Selects the type of clock to use. Valid options include:

- RT (monotonic real-time clock)
- CYC (cpu cycles measured with RDTSC)
- CPU (process-based CPU-time)

-cpu Dumps the cpu-time into the result file using the variable named $cpuTime.

***-csvOstep=value* *or* -csvOstep value** Value specifies csv-files for debug values for optimizer step.

-daeMode Enables daeMode simulation if the model was compiled with the omc flag --daeMode and ida method is used.

***-deltaXLinearize=value* *or* -deltaXLinearize value** Value specifies the delta x value for numerical differentiation used by linearization. The default value is sqrt(DBL_EPSILON*2e1).

***-deltaXSolver=value* *or* -deltaXSolver value** Value specifies the delta x value for numerical differentiation used by integration method. The default values is sqrt(DBL_EPSILON).

***-embeddedServer=value* *or* -embeddedServer value** Enables an embedded server. Valid values:

- none - default, run without embedded server
- opc-da - [broken] run with embedded OPC DA server (WIN32 only, uses proprietary OPC SC interface)
- opc-ua - [experimental] run with embedded OPC UA server (TCP port 4841 for now; will have its own configuration option later)
- filename - path to a shared object implementing the embedded server interface (requires access to internal OMC data-structures if you want to read or write data)

-embeddedServerPort=value ***or*** **-embeddedServerPort value** Value specifies the port number used by the embedded server. The default value is 4841.

-mat_sync=value ***or*** **-mat_sync value** Syncs the mat file header after emitting every N time-points.

-emit_protected Emits protected variables to the result-file.

-eps=value ***or*** **-eps value** Value specifies the number of convergence iteration to be performed for DataReconciliation

-f=value ***or*** **-f value** Value specifies a new setup XML file to the generated simulation code.

-help=value ***or*** **-help value** Get detailed information that specifies the command-line flag

For example, -help=f prints detailed information for command-line flag f.

-homAdaptBend=value ***or*** **-homAdaptBend value** Maximum trajectory bending to accept the homotopy step. Default: 0.5, which means the corrector vector has to be smaller than half of the predictor vector.

-homBacktraceStrategy=value ***or*** **-homBacktraceStrategy value** Value specifies the backtrace strategy in the homotopy corrector step. Valid values:

- fix - default, go back to the path by fixing one coordinate
- orthogonal - go back to the path in an orthogonal direction to the tangent vector

-homHEps=value ***or*** **-homHEps value** Tolerance respecting residuals for the homotopy H-function (default: 1e-5).

In the last step (lambda=1) newtonFTol is used as tolerance.

-homMaxLambdaSteps=value ***or*** **-homMaxLambdaSteps value** Maximum lambda steps allowed to run the homotopy path (default: system size * 100).

-homMaxNewtonSteps=value ***or*** **-homMaxNewtonSteps value** Maximum newton steps in the homotopy corrector step (default: 20).

-homMaxTries=value ***or*** **-homMaxTries value** Maximum number of tries for one homotopy lambda step (default: 10).

-homNegStartDir Start to run along the homotopy path in the negative direction.

If one direction fails, the other direction is always used as fallback option.

-homotopyOnFirstTry If the model contains the homotopy operator, directly use the homotopy method to solve the initialization problem. Without this flag, the solver first tries to solve the initialization problem without homotopy and only uses homotopy as fallback option.

-homTauDecFac=value ***or*** **-homTauDecFac value** Decrease homotopy step size tau by this factor if tau is too big in the homotopy corrector step (default: 10.0).

-homTauDecFacPredictor=value ***or*** **-homTauDecFacPredictor value** Decrease homotopy step size tau by this factor if tau is too big in the homotopy predictor step (default: 2.0).

-homTauIncFac=value ***or*** **-homTauIncFac value** Increase homotopy step size tau by this factor if tau can be increased after the homotopy corrector step (default: 2.0).

-homTauIncThreshold=value ***or*** **-homTauIncThreshold value** Increase the homotopy step size tau if homAdaptBend/bend > homTauIncThreshold (default: 10).

-homTauMax=value ***or*** **-homTauMax value** Maximum homotopy step size tau for the homotopy process (default: 10).

-homTauMin=value ***or*** **-homTauMin value** Minimum homotopy step size tau for the homotopy process (default: 1e-4).

-homTauStart=value ***or*** **-homTauStart value** Homotopy step size tau at the beginning of the homotopy process (default: 0.2).

-idaMaxErrorTestFails=value ***or*** **-idaMaxErrorTestFails value** Value specifies the maximum number of error test failures in attempting one step. The default value is 7.

-idaMaxNonLinIters=value ***or*** **-idaMaxNonLinIters value** Value specifies the maximum number of nonlinear solver iterations at one step. The default value is 3.

-idaMaxConvFails=value ***or*** **-idaMaxConvFails value** Value specifies the maximum number of nonlinear solver convergence failures at one step. The default value is 10.

-idaNonLinConvCoef=value ***or*** **-idaNonLinConvCoef value** Value specifies the safety factor in the nonlinear convergence test. The default value is 0.33.

-idaLS=value ***or*** **-idaLS value** Value specifies the linear solver of the ida integration method. Valid values:

- dense (ida internal dense method.)
- klu (ida use sparse direct solver KLU. (default))
- spgmr (ida generalized minimal residual method. Iterative method)
- spbcg (ida Bi-CGStab. Iterative method)
- sptfqmr (ida TFQMR. Iterative method)

-idaScaling Enable scaling of the IDA solver.

-idaSensitivity Enables sensitivity analysis with respect to parameters if the model is compiled with omc flag --calculateSensitivities.

-ignoreHideResult Emits also variables with HideResult=true annotation.

-iif=value ***or*** **-iif value** Value specifies an external file for the initialization of the model.

-iim=value ***or*** **-iim value** Value specifies the initialization method. Following options are available: 'symbolic' (default) and 'none'.

- none (sets all variables to their start values and skips the initialization process)
- symbolic (solves the initialization problem symbolically - default)

-iit=value ***or*** **-iit value** Value [Real] specifies a time for the initialization of the model.

-ils=value ***or*** **-ils value** Value specifies the number of steps for homotopy method (required: -iim=symbolic). The value is an Integer with default value 4.

-impRKOrder=value or **-impRKOrder value** Value specifies the integration order of the implicit Runge-Kutta method. Valid values: 1 to 6. Default order is 5.

-impRKLS=value or **-impRKLS value** Selects the linear solver of the integration methods impeuler, trapezoid and imprungekuta:

- iterativ - default, sparse iterativ linear solver with fallback case to dense solver
- dense - dense linear solver, SUNDIALS default method

-initialStepSize=value or **-initialStepSize value** Value specifies an initial step size, used by the methods: dassl, ida

-csvInput=value or **-csvInput value** Value specifies an csv-file with inputs for the simulation/optimization of the model

-exInputFile=value or **-exInputFile value** Value specifies an external file with inputs for the simulation/optimization of the model.

-stateFile=value or **-stateFile value** Value specifies an file with states start values for the optimization of the model.

-inputPath=value or **-inputPath value** Value specifies a path for reading the input files i.e., model_init.xml and model_info.json

-ipopt_hesse=value or **-ipopt_hesse value** Value specifies the hessematrix for Ipopt(OMC, BFGS, const).

-ipopt_init=value or **-ipopt_init value** Value specifies the initial guess for optimization (sim, const).

-ipopt_jac=value or **-ipopt_jac value** Value specifies the Jacobian for Ipopt(SYM, NUM, NUMDENSE).

-ipopt_max_iter=value or **-ipopt_max_iter value** Value specifies the max number of iteration for ipopt.

-ipopt_warm_start=value or **-ipopt_warm_start value** Value specifies lvl for a warm start in ipopt: 1,2,3,...

-jacobian=value or **-jacobian value** Select the calculation method for Jacobian used by the integration method:

- coloredNumerical (Colored numerical Jacobian, which is default for dassl and ida. With option -idaLS=klu a sparse matrix is used.)
- internalNumerical (Dense solver internal numerical Jacobian.)
- coloredSymbolical (Colored symbolical Jacobian. Needs omc compiler flag --generateSymbolicJacobian. With option -idaLS=klu a sparse matrix is used.)
- numerical (Dense numerical Jacobian.)
- symbolical (Dense symbolical Jacobian. Needs omc compiler flag --generateSymbolicJacobian.)

-jacobianThreads=value or **-jacobianThreads value** Value specifies the number of threads for jacobian evaluation in dassl or ida. The value is an Integer with default value 1.

-l=value or **-l value** Value specifies a time where the linearization of the model should be performed.

-l_datarec Emit data recovery matrices with model linearization.

-logFormat=value or **-logFormat value** Value specifies the log format of the executable:

- text (default)

- xml
- xmltcp (required -port flag)

-ls=value ***or*** **-ls value** Value specifies the linear solver method

- lapack (method using LAPACK LU factorization)
- lis (method using iterative solver Lis)
- klu (method using KLU sparse linear solver)
- umfpack (method using UMFPACK sparse linear solver)
- totalpivot (method using a total pivoting LU factorization for underdetermination systems)
- default (default method - LAPACK with total pivoting as fallback)

-ls_ipopt=value ***or*** **-ls_ipopt value** Value specifies the linear solver method for Ipopt, default mumps. Note: Use if you build ipopt with other linear solver like ma27

-lss=value ***or*** **-lss value** Value specifies the linear sparse solver method

- default (the default sparse linear solver (or a dense solver if there is none available))
- lis (method using iterative solver Lis)
- klu (method using klu sparse linear solver)
- umfpack (method using umfpack sparse linear solver)

-lssMaxDensity=value ***or*** **-lssMaxDensity value** Value specifies the maximum density for using a linear sparse solver. The value is a Double with default value 0.2.

-lssMinSize=value ***or*** **-lssMinSize value** Value specifies the minimum system size for using a linear sparse solver. The value is an Integer with default value 4001.

-lv=value ***or*** **-lv value** Value (a comma-separated String list) specifies which logging levels to enable. Multiple options can be enabled at the same time.

- stdout (this stream is always active, can be disabled with -lv=-stdout)
- assert (this stream is always active, can be disabled with -lv=-assert)
- LOG_DASSL (additional information about dassl solver)
- LOG_DASSL_STATES (outputs the states at every dassl call)
- LOG_DEBUG (additional debug information)
- LOG_DSS (outputs information about dynamic state selection)
- LOG_DSS_JAC (outputs jacobian of the dynamic state selection)
- LOG_DT (additional information about dynamic tearing)
- LOG_DT_CONS (additional information about dynamic tearing (local and global constraints))
- LOG_EVENTS (additional information during event iteration)
- LOG_EVENTS_V (verbose logging of event system)

- LOG_INIT (additional information during initialization)
- LOG_IPOPT (information from Ipopt)
- LOG_IPOPT_FULL (more information from Ipopt)
- LOG_IPOPT_JAC (check jacobian matrix with Ipopt)
- LOG_IPOPT_HESSE (check hessian matrix with Ipopt)
- LOG_IPOPT_ERROR (print max error in the optimization)
- LOG_JAC (outputs the jacobian matrix used by dassl)
- LOG_LS (logging for linear systems)
- LOG_LS_V (verbose logging of linear systems)
- LOG_NLS (logging for nonlinear systems)
- LOG_NLS_V (verbose logging of nonlinear systems)
- LOG_NLS_HOMOTOPY (logging of homotopy solver for nonlinear systems)
- LOG_NLS_JAC (outputs the jacobian of nonlinear systems)
- LOG_NLS_JAC_TEST (tests the analytical jacobian of nonlinear systems)
- LOG_NLS_RES (outputs every evaluation of the residual function)
- LOG_NLS_EXTRAPOLATE (outputs debug information about extrapolate process)
- LOG_RES_INIT (outputs residuals of the initialization)
- LOG_RT (additional information regarding real-time processes)
- LOG_SIMULATION (additional information about simulation process)
- LOG_SOLVER (additional information about solver process)
- LOG_SOLVER_V (verbose information about the integration process)
- LOG_SOLVER_CONTEXT (context information during the solver process)
- LOG_SOTI (final solution of the initialization)
- LOG_STATS (additional statistics about timer/events/solver)
- LOG_STATS_V (additional statistics for LOG_STATS)
- LOG_SUCCESS (this stream is always active, unless deactivated with -lv=-LOG_SUCCESS)
- LOG_UTIL (???)
- LOG_ZEROCROSSINGS (additional information about the zerocrossings)

***-mbi=value* or -mbi value** Value specifies the maximum number of bisection iterations for state event detection or zero for default behavior

***-mei=value* or -mei value** Value specifies the maximum number of event iterations. The value is an Integer with default value 20.

-maxIntegrationOrder=value ***or*** **-maxIntegrationOrder value** Value specifies maximum integration order, used by the methods: dassl, ida.

-maxStepSize=value ***or*** **-maxStepSize value** Value specifies maximum absolute step size, used by the methods: dassl, ida.

-measureTimePlotFormat=value ***or*** **-measureTimePlotFormat value** Value specifies the output format of the measure time functionality:

- svg
- jpg
- ps
- gif
- ...

-newtonFTol=value ***or*** **-newtonFTol value** Tolerance respecting residuals for updating solution vector in Newton solver. Solution is accepted if the (scaled) 2-norm of the residuals is smaller than the tolerance newtonFTol and the (scaled) newton correction (delta_x) is smaller than the tolerance newtonXTol. The value is a Double with default value 1e-12.

-newtonMaxStepFactor=value ***or*** **-newtonMaxStepFactor value** Maximum newton step factor mxnewtstep = maxStepFactor * norm2(xScaling). Used currently only by KINSOL.

-newtonXTol=value ***or*** **-newtonXTol value** Tolerance respecting newton correction (delta_x) for updating solution vector in Newton solver. Solution is accepted if the (scaled) 2-norm of the residuals is smaller than the tolerance newtonFTol and the (scaled) newton correction (delta_x) is smaller than the tolerance newtonXTol. The value is a Double with default value 1e-12.

-newton=value ***or*** **-newton value** Value specifies the damping strategy for the newton solver.

- damped (Newton with a damping strategy)
- damped2 (Newton with a damping strategy 2)
- damped_ls (Newton with a damping line search)
- damped_bt (Newton with a damping backtracking and a minimum search via golden ratio method)
- pure (Newton without damping strategy)

-nls=value ***or*** **-nls value** Value specifies the nonlinear solver:

- hybrid (Modification of the Powell hybrid method from minpack - former default solver)
- kinsol (SUNDIALS/KINSOL includes an interface to the sparse direct solver, KLU. See simulation option -nlsLS for more information.)
- newton (Newton Raphson - prototype implementation)
- mixed (Mixed strategy. First the homotopy solver is tried and then as fallback the hybrid solver.)
- homotopy (Damped Newton solver if failing case fixed-point and Newton homotopies are tried.)

-nlsInfo Outputs detailed information about solving process of non-linear systems into csv files.

-nlsLS=value ***or*** **-nlsLS value** Value specifies the linear solver used by the non-linear solver:

- default (chooses the nls linear solver based on which nls is being used.)
- totalpivot (internal total pivot implementation. Solve in some case even under-determined systems.)
- lapack (use external LAPACK implementation.)
- klu (use KLU direct sparse solver. Only with KINSOL available.)

-nlssMaxDensity=value ***or*** **-nlssMaxDensity value** Value specifies the maximum density for using a non-linear sparse solver. The value is a Double with default value 0.2.

-nlssMinSize=value ***or*** **-nlssMinSize value** Value specifies the minimum system size for using a non-linear sparse solver. The value is an Integer with default value 10001.

-noemit Do not emit any results to the result file.

-noEquidistantTimeGrid Output the internal steps given by dassl/ida instead of interpolating results into an equidistant time grid as given by stepSize or numberOfIntervals.

-noEquidistantOutputFrequency=value ***or*** **-noEquidistantOutputFrequency value** Integer value n controls the output frequency in noEquidistantTimeGrid mode and outputs every n-th time step

-noEquidistantOutputTime=value ***or*** **-noEquidistantOutputTime value** Real value timeValue controls the output time point in noEquidistantOutputTime mode and outputs every time>=k*timeValue, where k is an integer

-noEventEmit Do not emit event points to the result file.

-noRestart Disables the restart of the integration method after an event is performed, used by the methods: dassl, ida

-noRootFinding Disables the internal root finding procedure of methods: dassl and ida.

-noScaling Disables scaling for the variables and the residuals in the algebraic nonlinear solver KINSOL.

-noSuppressAlg Flag to not suppress algebraic variables in the local error test of the ida solver in daeMode. In general, the use of this option is discouraged when solving DAE systems of index 1, whereas it is generally encouraged for systems of index 2 or more.

-optDebugJac=value ***or*** **-optDebugJac value** Value specifies the number of iterations from the dynamic optimization, which will be debugged, creating .csv and .py files.

-optimizerNP=value ***or*** **-optimizerNP value** Value specifies the number of points in a subinterval. Currently supports numbers 1 and 3.

-optimizerTimeGrid=value ***or*** **-optimizerTimeGrid value** Value specifies external file with time points.

-output=value ***or*** **-output value** Output the variables a, b and c at the end of the simulation to the standard output: time = value, a = value, b = value, c = value

-outputPath=value ***or*** **-outputPath value** Value specifies a path for writing the output files i.e., model_res.mat, model_prof.intdata, model_prof.realdata etc.

-override=value ***or*** **-override value** Override the variables or the simulation settings in the XML setup file For example: var1=start1,var2=start2,par3=start3,startTime=val1,stopTime=val2

-overrideFile=value ***or*** **-overrideFile value** Will override the variables or the simulation settings in the XML setup file with the values from the file. Note that: -overrideFile CANNOT be used with -override. Use when variables for -override are too many. overrideFileName contains lines of the form: var1=start1

-port=value ***or*** **-port value** Value specifies the port for simulation status (default disabled).

-r=value ***or*** **-r value** Value specifies the name of the output result file. The default file-name is based on the model name and output format. For example: Model_res.mat.

-reconcile Run the DataReconciliation algorithm for constrained equation

-rt=value ***or*** **-rt value** Value specifies the scaling factor for real-time synchronization (0 disables). A value > 1 means the simulation takes a longer time to simulate.

-s=value ***or*** **-s value** Value specifies the integration method. For additional information see the *User's Guide*

- euler - Euler - explicit, fixed step size, order 1
- heun - Heun's method - explicit, fixed step, order 2
- rungekutta - classical Runge-Kutta - explicit, fixed step, order 4
- impeuler - Euler - implicit, fixed step size, order 1
- trapezoid - trapezoidal rule - implicit, fixed step size, order 2
- imprungekutta - Runge-Kutta methods based on Radau and Lobatto IIA - implicit, fixed step size, order 1-6(selected manually by flag -impRKOrder)
- irksco - own developed Runge-Kutta solver - implicit, step size control, order 1-2
- dassl - default solver - BDF method - implicit, step size control, order 1-5
- ida - SUNDIALS IDA solver - BDF method with sparse linear solver - implicit, step size control, order 1-5
- rungekuttaSsc - Runge-Kutta based on Novikov (2016) - explicit, step size control, order 4-5 [experimental]
- symSolver - symbolic inline Solver [compiler flag +symSolver needed] - fixed step size, order 1
- symSolverSsc - symbolic implicit Euler with step size control [compiler flag +symSolver needed] - step size control, order 1
- qss - A QSS solver [experimental]
- optimization - Special solver for dynamic optimization

-single Output results in single precision (mat-format only).

-steps Dumps the number of integration steps into the result file.

-steadyState Aborts the simulation if steady state is reached.

-steadyStateTol=value ***or*** **-steadyStateTol value** This relative tolerance is used to detect steady state: max(|d(x_i)/dt|/nominal(x_i)) < steadyStateTol

-sx=value ***or*** **-sx value** Value specifies an csv-file with inputs as covariance matrix Sx for DataReconciliation

***-keepHessian=value* or -keepHessian value** Value specifies the number of steps, which keep Hessian matrix constant.

-w Shows all warnings even if a related log-stream is inactive.

第26章　Technical Details

This chapter gives an overview of some implementation details that might be interesting when building tools around OpenModelica.

26.1 The MATv4 Result File Format

The default result-file format of OpenModelica is based on MATLAB level 4 MAT-files as described in the MATLAB documentation. This format can be read by tools such as MATLAB, Octave, Scilab, and SciPy. OpenModelica will write the result-files in a particular way that can be read by tools such as DyMat and Dymola (OpenModelica can also read files generated by Dymola since the used format is the same).

The variables stored in the MAT-file are (in the order required by OpenModelica):

Aclass

- `Aclass(1,:)` is always `Atrajectory`
- `Aclass(2,:)` is `1.1` in OpenModelica
- `Aclass(3,:)` is empty
- `Aclass(4,:)` is either `binTrans` or `binNormal`

The most important part of the variable is `Aclass(4,:)` since there are two main ways the result-file is stored: transposed or not. For efficiency, the result-file is written time-step by time-step during simulation. But the best way to read the data for a single variable is if the variables are stored variable by variable. If `Aclass(4,:)` is `binTrans`, all matrices need to be transposed since the file was not transposed for efficient reading of the file. Note that this affects all matrices, even matrices that do not change during simulation (such as name and description).

name Is an n x m character (int8) matrix, where n is the number of variables stored in the result-file (including time). m is the length of the longest variable. OpenModelica stores the trailing part of the name as NIL bytes (0) whereas other tools use spaces for the trailing part.

description Is an n x m character (int8) matrix containing the comment-string corresponding to the variable in the name matrix.

dataInfo Is an n x 4 integer matrix containing information for each variable (in the same order as the name and description matrices).

- `dataInfo(i,1)` is `1` or `2`, saying if variable i is stored in the data_1 or data_2 matrix. If it is `0`, it is the abscissa (time variable).

- `dataInfo(i,2)` contains the index in the data_1 or data_2 matrix. The index is 1-based and may contain several variables pointing to the same row (alias variables). A negative value means that the variable is a negated alias variable.
- `dataInfo(i,3)` is 0 to signify linear interpolation. In other tools the value is the number of times differentiable this variable is, which may improve plotting.
- `dataInfo(i,4)` is `-1` in OpenModelica to signify that the value is not defined outside the time range. `0` keeps the first/last value when going outside the time range and `1` performs linear interpolation on the first/last two points.

data_1 If it is an n x 1 matrix it contains the values of parameters. If it is an n x 2 matrix, the first and second column signify start and stop-values.

data_2 Each row contains the values of a variable at the sampled times. The corresponding time stamps are stored in `data_2(1,:)`. `data_2(2,1)` is the value of some variable at time `data_2(1,1)`.

第27章　DataReconciliation

The objective of data reconciliation is to use physical models to decrease measurement uncertainties on physical quantities. Data reconciliation is possible only when redundant measurements are available for a given physical quantity.

27.1 Defining DataReconciliation Problem in OpenModelica

To define DataReconciliation Problem in OpenModelica, The Modelica model must be defined with the following

- The list of variables of interest, which is defined in the modelica model as a special variable attribute (uncertain=Uncertainty.refine)
- The list of approximated equations. which is defined in the modelica model as a special annotation (__OpenModelica_ApproximatedEquation=true)

The list of Variable of interest are mandatory and the list of approximated equations are optional. An example of modelica model with dataReconciliation problem is given below,

```
model Splitter1
  Real Q1(uncertain=Uncertainty.refine);
  Real Q2(uncertain=Uncertainty.refine);
  Real Q3(uncertain=Uncertainty.refine);
  parameter Real P01 =3;
  parameter Real P02 =1;
  parameter Real P03 =1;
  Real T1_P1, T1_P2, T2_P1, T2_P2, T3_P1, T3_P2;
  Real V_Q1, V_Q2, V_Q3;
  Real T1_Q1, T1_Q2, T2_Q1, T2_Q2, T3_Q1, T3_Q2;
  Real P, V_P1, V_P2, V_P3;
equation
  T1_P1 = P01;
  T2_P2 = P02;
  T3_P2 = P03;
  T1_P1 - T1_P2 = Q1^2 annotation (__OpenModelica_ApproximatedEquation=true);
  T2_P1 - T2_P2 = Q2^2 annotation (__OpenModelica_ApproximatedEquation=true);
  T3_P1 - T3_P2 = Q3^2 annotation (__OpenModelica_ApproximatedEquation=true);
  V_Q1 = V_Q2 + V_Q3;
  V_Q1 = T1_Q2;
  T1_Q2 = Q1;
  V_Q2 = T2_Q1;
  T2_Q1 = Q2;
  V_Q3 = T3_Q1;
  T3_Q1 = Q3;
  T1_P2 = V_P1;
  V_P1 = P;
  T2_P1 = V_P2;
  V_P2 = P;
```

(次のページに続く)

(前のページからの続き)

```
  T3_P1 = V_P3;
  V_P3 = P;
  T1_Q1 = Q1;
  T2_Q2 = Q2;
  T3_Q2 = Q3;
end Splitter1;
```

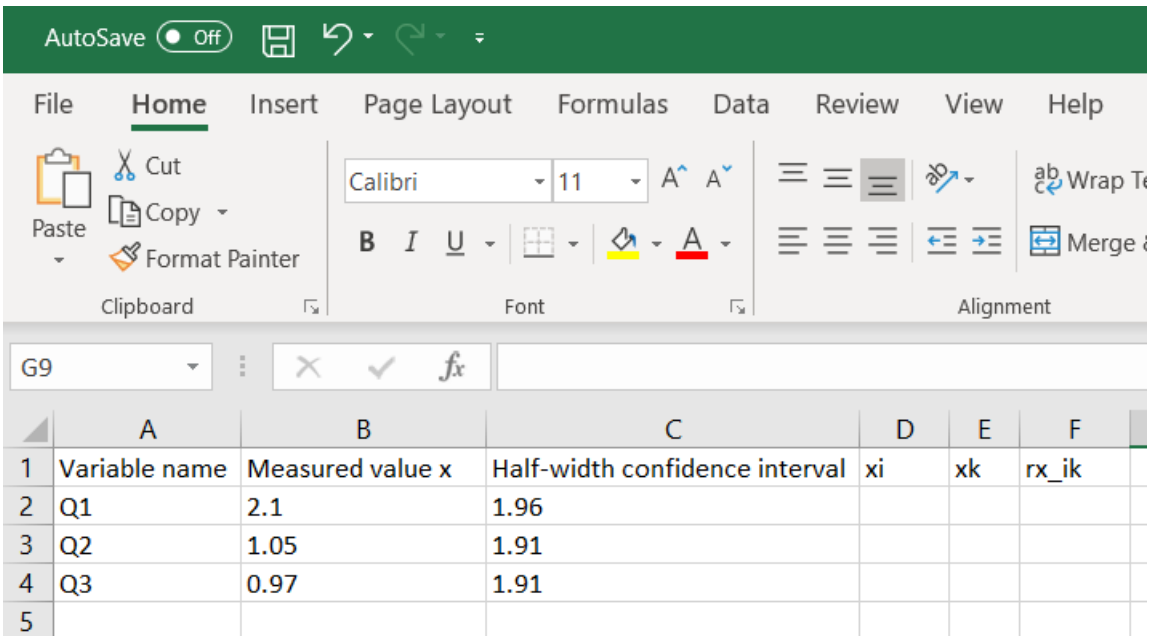
After defining the modelica model, the users must define the dataReconciliation Input File.

27.1.1 DataReconciliationInputFile

The dataReconciliation Input file is a csv file with the the following headers,

- Variable Names - names of the Uncertainty variables, given in the modelica model
- Measured Value-x – Values given by the users
- HalfWidthConfidenceInterval – Values given by the users, which computes Covariance Matrix Sx
- xi – co-relation- coefficients
- xk - co-relation- coefficients
- rx_ik- value associated with co-relation coefficients

The first 3 column, Variable Names, Measured Value-x and HalfWidthConfidenceInterval are mandatory The remaining column xi, xk, rx_ik are correlation-coefficients which are optional. An example csv file is given below



	A	B	C	D	E	F
1	Variable name	Measured value x	Half-width confidence interval	xi	xk	rx_ik
2	Q1	2.1	1.96			
3	Q2	1.05	1.91			
4	Q3	0.97	1.91			
5						

Figure 27.1: An example DataReconciliationInput file(.csv)

The ordering of variables in the csv files should be defined in correct order on how it is declared in the model, for example in the above example we have uncertain variables defined in the following order Q1,Q2 and Q3

and the same order should be followed for the csv file in order to match the jacobian columns generated for dataReconciliation Otherwise the dataReconciliation procedure computes wrong results.

Now we are ready to run the DataReconciliation procedure in OpenModelica.

27.2 DataReconciliation Support with Scripting Interface

The data Reconciliation procedure is possible to run through OpenModelica scripting interface(.mos file). An example mos script (a.mos) is present below.

```
setCommandLineOptions("--preOptModules+=dataReconciliation");
getErrorString();
loadFile("DataReconciliationSimpleTests/package.mo");
getErrorString();
simulate(DataReconciliationTests.Splitter1,simflags="-reconcile -sx=./Splitter1_Sx.
↪csv -eps=0.0023 -lv=LOG_JAC");
getErrorString();
```

To start the dataReconciliation procedure via command line interface, the users have to enable the dataReconciliation module which is done via setCommandLineOptions("--preOptModules+=dataReconciliation") which runs the extraction algorithm for dataReconciliation procedure. And finally the users must specify 3 runtime simulation flags given below

1. reconcile – runtime flag which starts the dataReconciliation Procedure
2. sx – csv file Input
3. eps – small value given by users

The Flag -lv=LOG_JAC is optional and can be used for debugging.

And finally run the mos script(a.mos) with omc

>> omc a.mos

The HTML Reports, the Csv files and the debugging log are generated in the current directory see *DataReconcilation Results*.

27.3 DataReconciliation Support in OMEdit

The DataReconciliation setup can be launched by,

- Selecting Simulation > Simulation Setup from the menu. (requires a model to be active in ModelWidget)
- Clicking on the Simulation Setup toolbar button. (requires a model to be active in ModelWidget)
- Right clicking the model from the Libraries Browser and choosing Simulation Setup.

27.3.1 TranslationFlag Tab

From the translationFlag tab, do the following,

- check the Enable dataReconciliation checkbox.

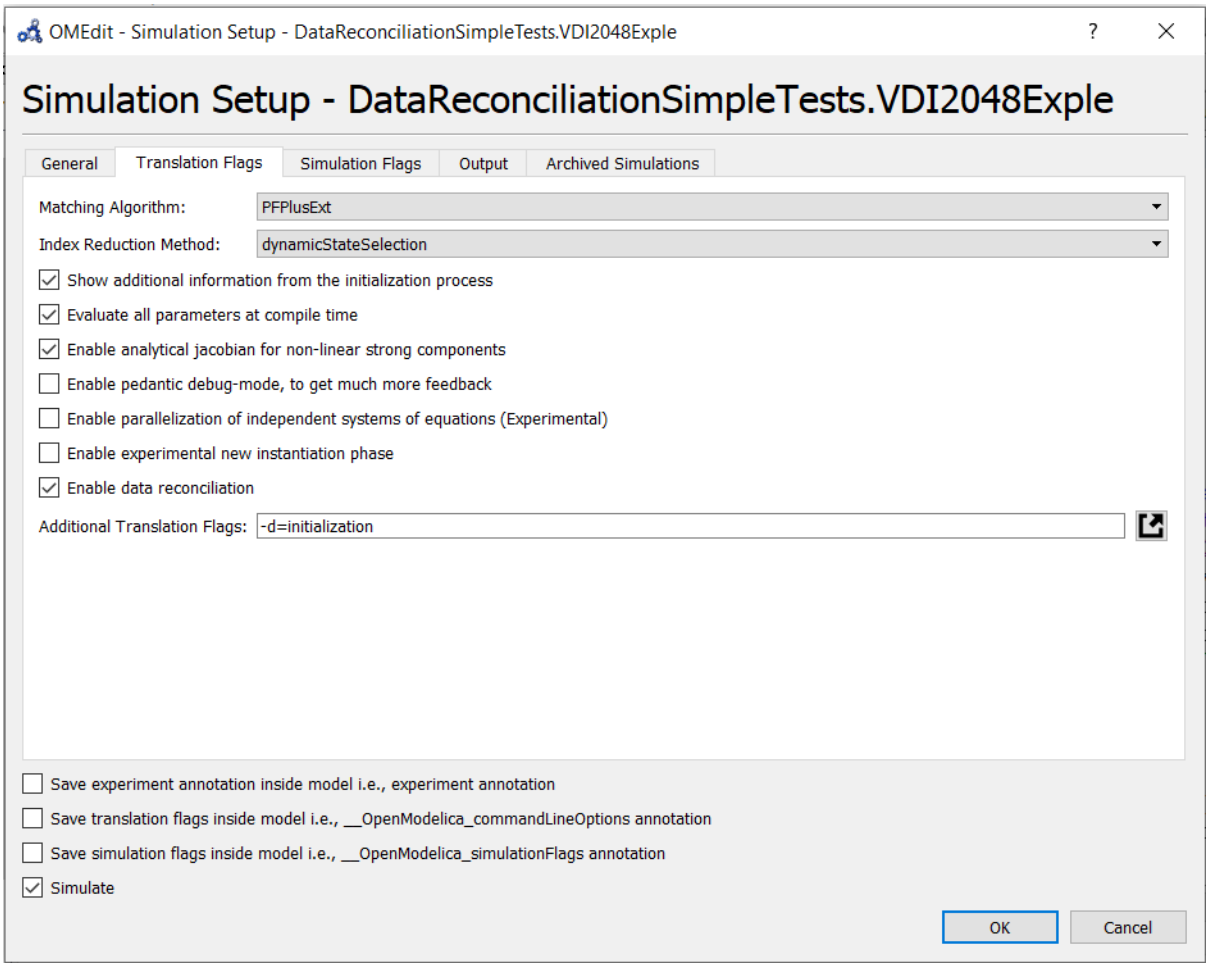


Figure 27.2: Setting DataReconciliation TraslationFlag

27.3.2 SimulationFlag Tab

From the SimulationFlag tab, do the following,

- check the DataReconciliation Algorithm for Constrained Equation checkbox.
- load the input file with dataReconciliation inputs, only csv file is accepted.
- fill in the Epsilon value (e.g) 0.001

And finally press the ok button to start the dataReconciliation procedure

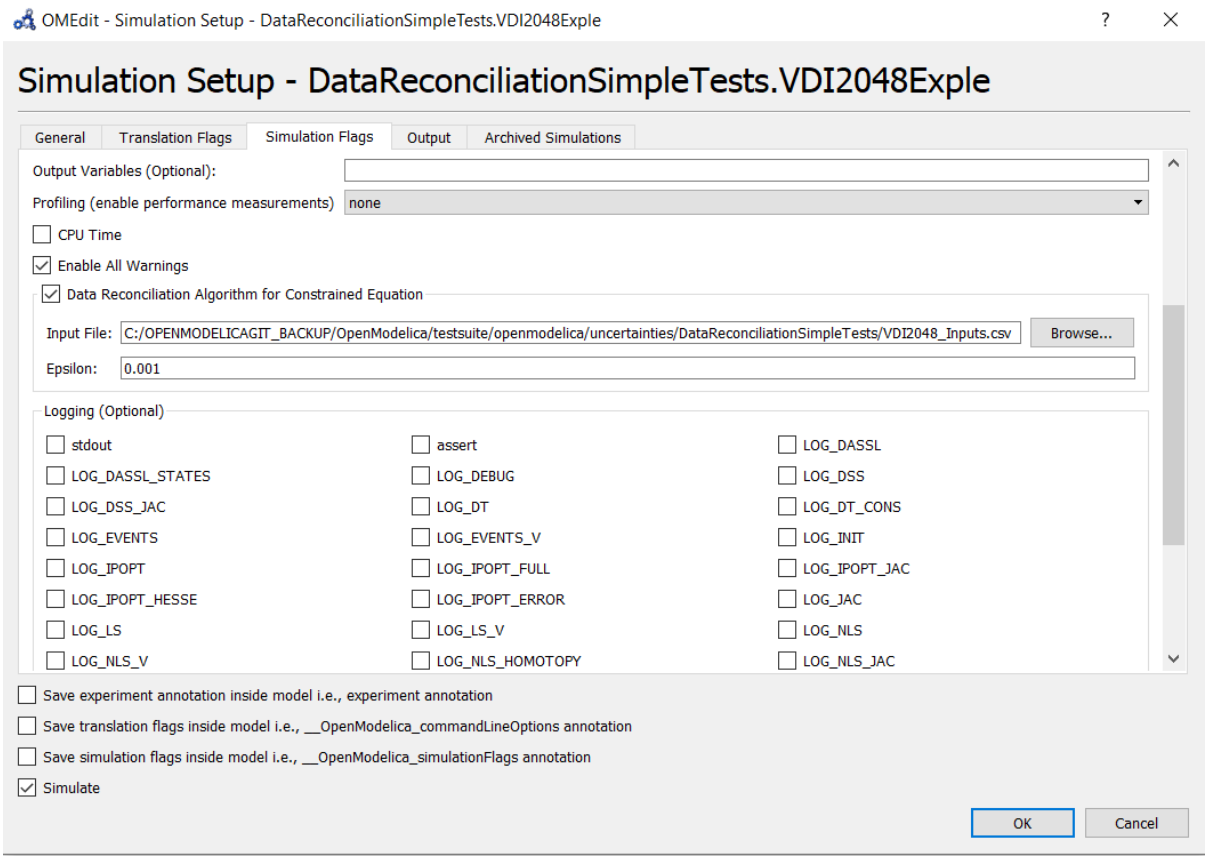


Figure 27.3: Setting DataReconciliation SimuationFlag

27.3.3 Generating the InputFile and Running the DataReconciliation

Generating an empty csv file with variable names makes it easy for the users to fill in the datas, so that ordering of variables and names are not mismatched. This is an important step as variable ordering should match with the jacobian columns generated for dataReconciliation procedure. The input file is named as "modelname_Inputs.csv" which is generated in the current working directory of the model. This step shall be done for the first time and the next time when running the dataReconciliation for the same model, we can directly set the input file and run the DataReconciliation procedure.

This is done in 2 steps.

- Setting the TranslationFlag defined in *TranslationFlag Tab*. and press the Ok button.

And then from the plotting window variable browser, right click on the model and select the "re-simulate Setup" as shown below

Which opens the simulation set-up dialog window and select the simulation Flag tab defined in *SimulationFlag Tab*. and load the csv file and fill in the epsilon value and press the "Ok" button to start the Data Reconciliation Procedure.

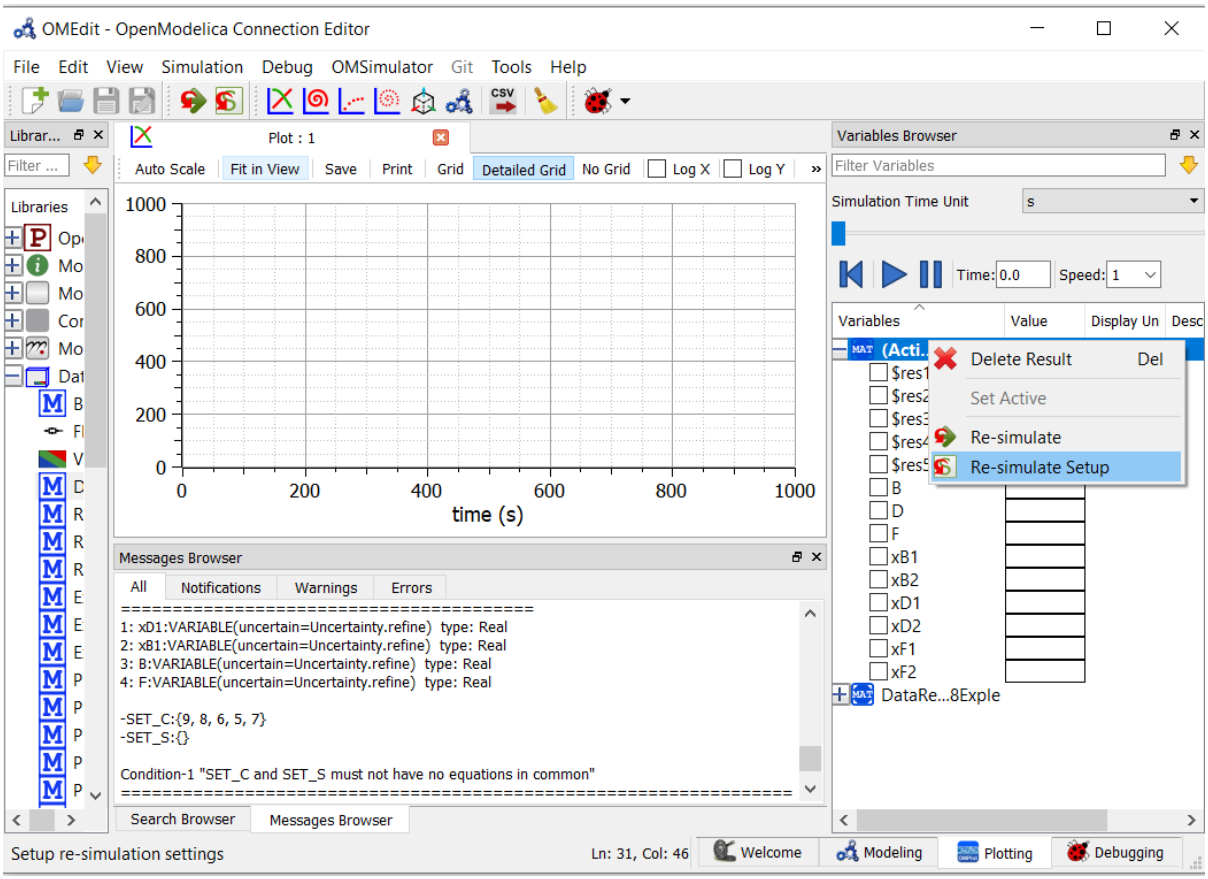


Figure 27.4: Select the re-simulate setup

27.4 DataReconcilation Results

After the Data Reconciliation procedure is completed, the results are generated in the working directory. The default working directory in OMEdit is set to local temp directory of the operating system. The users can change the working directory of OMEdit by, Tools > Options > General > WorkingDirectory

A separate working directory is created in the working directory. The directory is named based on the modelName and the result files are stored in that directory. Two result files are generated namely.

- HTML Report.
- CSV file

An Example of Result directory is given below,

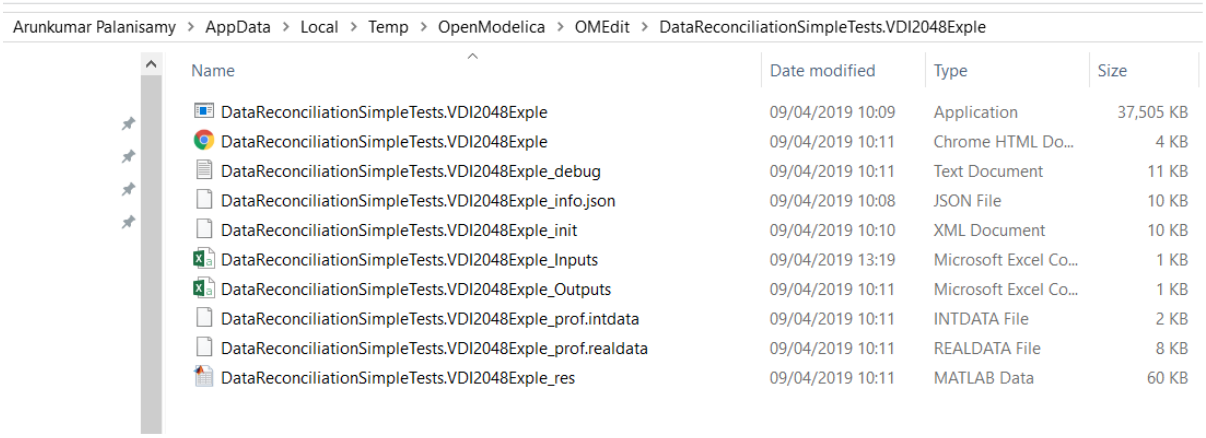


Figure 27.5: Result Directory Structure

27.4.1 HTML Report

The html report is named with modelname.html. The Html report contains 3 section namely 1. Overview 2. Analysis and 3. Results

The Overview section provides the general details of the model such as Modelicafile, ModelName, ModelDirectory, InputFiles and Generated Date and Time of the Report.The Analysis section provides information about the data Reconciliation procedure such as Number of Extracted equations in setC, Number of variable to be Reconciled which are Variable of interest, Number of Iterations to Converge, Final Converged Value ,Epsilon value provided by the users and Results of Global test.

The Results section provides the numerical values computed by the data Reconciliation algorithm. The table contains 8 columns namely,

1. Variables to be Reconciled – names of the Uncertainty variables, given in the modelica model
2. Initial Measured Values – numerical values given by the users
3. Reconciled Values – Calculated values according to Data Reconciliation Procedure.

4. Initial Uncertainty Values – Half Width confidence interval provides by the users, which is later used to compute the Covariance Matrix Sx.
5. Reconciled Uncertainty Values – Calculated Values according to Data Reconciliation Procedure.
6. Results of Local Tests – Calculated values according to Data Reconciliation Procedure
7. Values of Local Tests – Calculated values according to Data Reconciliation Procedure
8. Margin to correctness – Calculated values according to Data Reconciliation Procedure

A sample HTML Report generated for Splitter1.mo model is presented below.

DataReconciliation Report

Overview:

ModelFile: DataReconciliationSimpleTests.Splitter1.mo
ModelName: DataReconciliationSimpleTests.Splitter1
ModelDirectory: C:/OPENMODELICAGIT/OpenModelica/testsuite/openmodelica/uncertainties/DataReconciliationSimpleTests
Measurement Files: C:/OPENMODELICAGIT_BACKUP/OpenModelica/testsuite/openmodelica/uncertainties/DataReconciliationSimpleTests/Splitter1Inputs.csv
Generated: Wed Apr 10 12:08:45 2019 by **OpenModelica-v1.14.0-dev-225-g199705757 (64-bit)**

Analysis:

Number of Extracted equations: 1
Number of Variables to be Reconciled: 3
Number of Iteration to Converge: 2
Final Converged Value(J*/r) : 0
Epsilon : 0.001
Final Value of the objective Function (J*) : 0
Chi-square value : 3.84146
Result of Global Test : TRUE

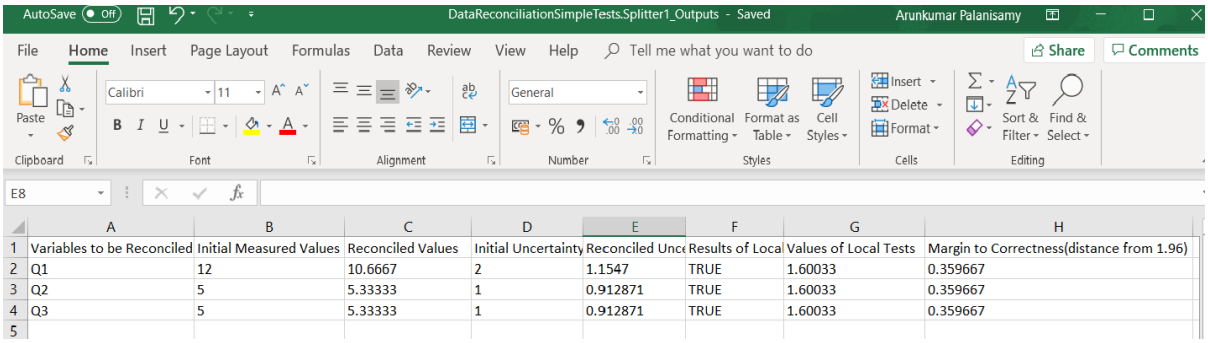
Results:

Variables to be Reconciled	Initial Measured Values	Reconciled Values	Initial Uncertainty Values	Reconciled Uncertainty Values	Results of Local Tests	Values of Local Tests	Margin to Correctness(distance from 1.96)
Q1	12	10.6667	2	1.1547	TRUE	1.60033	0.359667
Q2	5	5.33333	1	0.912871	TRUE	1.60033	0.359667
Q3	5	5.33333	1	0.912871	TRUE	1.60033	0.359667

Figure 27.6: HTML Report

27.4.2 Csv file

Along with the Html Report, an output csv file is also generated which mainly contains the Results section of the HTMl report in a csv format. The csv file is named with modelname_Outputs.csv. An example output csv file is presented below.



	A	B	C	D	E	F	G	H
1	Variables to be Reconciled	Initial Measured Values	Reconciled Values	Initial Uncertainty	Reconciled Unce	Results of Local	Values of Local Tests	Margin to Correctness(distance from 1.96)
2	Q1	12	10.6667	2	1.1547	TRUE	1.60033	0.359667
3	Q2	5	5.33333	1	0.912871	TRUE	1.60033	0.359667
4	Q3	5	5.33333	1	0.912871	TRUE	1.60033	0.359667
5								

Figure 27.7: Output Csv file

27.4.3 Logging and Debugging

All the Computations of data Reconciliation procedure are logged into log file. The log file is named as modelname_debug.log. For Detailed Debugging the flag LOG_JAC checkbox can be checked see *SimulationFlag Tab*.

第28章　Frequently Asked Questions (FAQ)

Below are some frequently asked questions in three areas, with associated answers.

28.1 OpenModelica General

- **Q: OpenModelica does not read the MODELICAPATH environment variable,** even though this is part of the Modelica Language Specification.
- **A: Use the OPENMODELICALIBRARY environment variable instead. We have** temporarily switched to this variable, in order not to interfere with other Modelica tools which might be installed on the same system. In the future, we might switch to a solution with a settings file, that also allows the user to turn on the MODELICAPATH functionality if desired.
- **Q: How do I enter multi-line models into OMShell since it evaluates** when typing the Enter/Return key?
- **A: There are basically three methods: 1) load the model from a file** using the pull-down menu or the loadModel command. 2) Enter the model/function as one (possibly long) line. 3) Type in the model in another editor, where using multiple lines is no problem, and copy/paste the model into OMShell as one operation, then push Enter. Another option is to use OMNotebook instead to enter and evaluate models.

28.2 OMNotebook

- Q: OMNotebook hangs, what to do?
- **A: It is probably waiting for the omc.exe (compiler) process. (Under** windows): Kill the processes omc.exe, g++.exe (C-compiler), as.exe (assembler), if present. If OMNotebook then asks whether to restart OMC, answer yes. If not, kill the process OMNotebook.exe and restart manually.
- **Q: After a previous session, when starting OMNotebook again, I get a** strange message.
- **A: You probably quit the previous OpenModelica session in the wrong** way, which left the process omc.exe running. Kill that process, and try starting OMNotebook again.
- **Q: I copy and paste a graphic figure from Word or some other** application into OMNotebook, but the graphic does not appear. What is wrong?
- **A: OMNotebook supports the graphic picture formats supported by Qt 4,** including the .png, .bmp (bitmap) formats, but not for example the gif format. Try to convert your picture into one of the

supported formats, (e.g. in Word, first do paste as bitmap format), and then copy the converted version into a text cell in OMNotebook.

- **Q: I select a cell, copy it (e.g. Ctrl-C), and try to paste it at** another place in the notebook. However, this does not work. Instead some other text that I earlier put on the clipboard is pasted into the nearest text cell.
- **A: The problem is wrong choice of cursor mode, which can be text** insertion or cell insertion. If you click inside a cell, the cursor become vertical, and OMNotebook expects you to paste text inside the cell. To paste a cell, you must be in cell insertion mode, i.e., click between two cells (or after a cell), you will get a vertical line. Place the cursor carefully on that vertical line until you see a small horizontal cursor. Then you should past the cell.
- **Q: I am trying to click in cells to place the vertical character** cursor, but it does not seem to react.
- **A: This seems to be a Qt feature. You have probably made a selection** (e.g. for copying) in the output section of an evaluation cell. This seems to block cursor position. Click again in the output section to disable the selection. After that it will work normally.
- **Q: I have copied a text cell and start writing at the beginning of** the cell. Strangely enough, the font becomes much smaller than it should be.
- **A: This seems to be a Qt feature. Keep some of the old text and start** writing the new stuff inside the text, i.e., at least one character position to the right. Afterwards, delete the old text at the beginning of the cell.

28.3 OMDev - OpenModelica Development Environment

- **Q: I get problems compiling and linking some files when using OMDev** with the MINGW (Gnu) C compiler under Windows.
- **A: You probably have some Logitech software installed. There is a** known bug/incompatibility in Logitech products. For example, if lvprcsrv.exe is running, kill it and/or prevent it to start again at reboot; it does not do anything really useful, not needed for operation of web cameras or mice.

第29章　Major OpenModelica Releases

This Appendix lists the most important OpenModelica releases and a brief description of their contents. Right now versions from 1.3.1 to 2.0.0 are described.

29.1 Release Notes for OpenModelica 2.0.0

29.1.1 OpenModelica Compiler (OMC)

29.1.2 Graphic Editor OMEdit

29.1.3 FMI Support

29.1.4 Other things

-,col=changelog,group=component,format=table)

29.2 Release Notes for OpenModelica 1.16.0

29.2.1 OpenModelica Compiler (OMC)

29.2.2 Graphic Editor OMEdit

29.2.3 FMI Support

29.2.4 Other things

-,col=changelog,group=component,format=table)

29.3 Release Notes for OpenModelica 1.15.0

29.3.1 OpenModelica Compiler (OMC)

29.3.2 Graphic Editor OMEdit

29.3.3 FMI Support

29.3.4 Other things

-,col=changelog,group=component,format=table)

29.4 Release Notes for OpenModelica 1.14.0

29.4.1 OpenModelica Compiler (OMC)

The most dramatic enhancement of the OpenModelica Compiler is the New Frontend, which on the average gives a factor of 10-20 speed improvement in the flattening phase of compilation. The new frontend is default in this release, but a feature is implemented that allows the user to switch to the old frontend if problems appear for a specific model The speed of the OMEdit GUI has only slightly increased in this version, since it is still dependent mostly on the old frontend. Further GUI speed increases are available in the coming OpenModelica.1.15.0. About 200 issues have been fixed, including enhancements, compared to the previous 1.13.2 release. The bug fixes are on trac.

OpenModelica Compiler New Frontend news: • Implementation of expandable connectors completed, a rather large piece of work. • A number of smaller enhancements and fixes • The New Frontend (NF)gives slightly better simulation coverage on MSL 3.2.3 than the Old Frontend • The New Frontend is on the average about 20 times faster on flattening. • Remaining work is mainly on further bug fixing and testing the new frontend for all other libraries, as well as more work on modifiers of arrays in conjunction with non-expanded arrays. (The array modifiers have now been implemented in 1.16.0 but not yet it 1.14.0 in order to not delay the 1.14.0 release)

29.4.2 Graphic Editor OMEdit

- Drag and drop for the text layer. • Auto completion of class names,

components and annotations. • GUI for data reconciliation – a method for increasing sensor data precision • Improved duplication functionality for copying classes. • Better handling of Compiler flags. • Partly completed: annotations for dynamic icon update. • Support for connectorSizing annotation • Several bug fixes. You can find the list here. • Docs: https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omedit.html . • Autocomplete annotations. • Support for Icon/Diagram map annotation • Copy paste functionality • Reset OMEdit settings/options. • Array plots update on re-simulation • Support for connectorSizing annotation. • Drag and drop class names to text layer in OMEdit • OMPlot: Improved plotting e.g., top and bottom margins for better view, snap to curve etc. • GUI support for replaceable libraries is being tested in a separate branch and will be made available in the coming 1.15.0 release.

29.4.3 OMC backend and run-time system

- A new more efficient and correct implementation of arrays and records.
- The FMI OMSimulator API calls are now also available in the OMC API

functions, e.g. for use from OMNotebook, Jupyter notebooks.

Backend new features

- Added possibility to generate analytic Jacobians for linear strong

components

`` * -Use flag LSanalyticJacobian to enable analytical Jacobian for linear loops. Default false.``

- Added output language options for linearization: matlab, python,

julia. • Available with --linearizationDumpLanguage=modelica/matlab/python/julia. Default is modelica.

Backend enhancements

- Unified Jacobian evaluation from DASSL and IDA integrators • Added

result check for linear solvers Lis, Klu, Total Pivot and Umfpack if a residual function is available. • Improved debug dumping

`` * -d=bltdump (Index reduction information)``
`` * -d=initialization``
`` * -d=dumpLoops``

- Improved warning for iteration variables:

`` * Only warn about non-linear iteration variables with default start attribute.``
`` * Other variables have no influence on simulation at all.``

- Build instructions for OpenModelica on Windows Subsystem for Linux •

Improved Jacobian evaluation with translation flag -d=symJacConstantSplit (requires --generateSymbolicJacobian) Generate Jacobians with separated constant part to split equations that are independent of the seed vector. These equations only need to be evaluated only once per Jacobian evaluation.

Backend bugfixes

- Homotopy: Use simplified version only during initialization to avoid

errors during matching and differentiation. • Logging for Homotopy path fixed so log can be loaded in OMEdit. • Support general function call differentiation for equations in residual form. • Equations in residual form don't fail during index reduction any more.

29.4.4 FMI Support

Bug fixes to FMI export, see below

29.4.5 Other things

-,col=changelog,group=component,format=table)

29.5 Release Notes for OpenModelica 1.13.0

- OMSimulator 2.0 – the second release of our efficient FMI Simulation

tool including a GUI for FMI Composition, co-simulation and model-exchange simulation, and SSP standard support. - Model and library encryption/decryption support. (Only for usage by OSMC member organizations) - Improved OpenModelica DAEMode for efficient solution of large Modelica models. - Julia scripting API to OpenModelica. - Basic Matlab scripting API to OpenModelica. - OMSysIdent - parameter estimation module for linear and non-linear parametric dynamic models. - Interactive simulation and control of simulations with OPC-UA. - PDEModelica1 - experimental support for one-dimensional PDEs in Modelica. - Analytic directional derivatives for FMI export and efficient calculation of multiple Jacobian columns – giving much faster simulation for some models - Enhanced OMEdit – including fast multi-file search. - Improved error messages and stability. - A version of the new fast compiler frontend available for testing, can be enabled by a flag Currently (December 10), simulates about 84% of MSL 3.2.2

Note: the replaceable GUI support has been moved to OpenModelica 1.14.0 and will be available in nightly builds.

-,col=changelog,group=component,format=table)

29.6 Release Notes for OpenModelica 1.12.0

- A new (stand-alone) FMI- and TLM-based simulation tool OMSimulator, first version for connected FMUs, TLM objects, Simulink models (via wrappers), Adams models (via wrappers), BEAST models (via wrappers), Modelica models
- Graphic configuration editing of composite models consisting of FMUs
- Basic graphical editing support for state machines and transitions

- Faster lookup processing, making some libraries faster to browse and compile
- Additional advanced visualization features for multibody animation
- Increased library coverage including significantly increased verification coverage
- Increased tool interoperability by addition of the ZeroMQ communications protocol
- Further enhanced OMPython including linearization, now also working with Python 3
- Support for RedHat/Fedora binary builds of OpenModelica

29.6.1 OpenModelica Compiler (OMC)

- Faster lookup processing
- Initializing external objects together with parameters
- Handle exceptions in numeric solvers
- Support for higher-index discrete clock partitions
- Improved unit checking
- Improved initialization of start values
- Decreased compilation time of models with large size arrays
- New approach for homotopy-based initialization (still experimental)
- A bunch of fixes: Bugs, regressions, performance issues
- Improved Dynamic Tearing by adding constraints for the casual set
- Improved module wrapFunctionCalls with one-time evaluation of Constant CSE-variables
- Added initOptModule for inlineHomotopy
- Added configuration flag tearingStrictness to influence solvability
- New methods for inline integration for continuous equations in clocked partitions, now covering: ExplicitEuler, ImplicitEuler, SemiImplicitEuler and ImplicitTrapezoid
- Complete implementation of synchronous features in C++ runtime
- Refactored linear solver of C++ runtime
- Improved Modelica_synchronous_cpp coverage
- New common linear solver module, optionally sparse, for the C++ runtime
- Coverage of most of the OpenHydraulics library
- Improved coverage of ThermoSysPro, IdealizedContact and Chemical libraries
- Support of time events for cpp-simulation and enabled time events in cpp-FMUs
- Global homotopy method for initialization

- Scripting API to compute accumulated errors (1-norm, 2-norm, max. error) of 2 time series

29.6.2 Graphic Editor OMEdit

- Additional advanced visualization features for multibody animation (transparency, textures, change colours by dialog)
- An HTML WYSIWYG Editor, e.g. useful for documentation
- Support for choices(checkBox=true) annotation.
- Support for loadSelector & saveSelector attribute of Dialog annotation.
- Panning of icon/diagram view and plot window.
- AutoComplete feature in text editing for keywords, types, common Modelica constructs
- Follow connector transformation from Diagram View to Icon View.
- Further stability improvements
- Improved performance for rendering some icons using the interactive API
- Improved handling of parameters that cannot be evaluated in Icon annotations
- Basic graphic editing support for state machines and transitions (not yet support for showing state internals on diagram layer)
- Interactive state manipulation for FMU-based animations

29.6.3 FMI Support

- A new (stand-alone) FMI- and TLM-based simulation tool OMSimulator, first version (a main deliverable of the OPENCPS project, significant contributions and code donations from SKF)
- Graphic configuration editing of composite models consisting of FMUs
- Co-simulation/simulation of connected FMUs, TLM objects, Simulink models (via wrappers), Adams models (via wrappers), BEAST models (via wrappers), Modelica models.

29.6.4 Other things

- Increased OpenModelica tool interoperability by adding the ZeroMQ communications protocol in addition to the previously available Corba. This also enables Python 3 usage in OMPython on all platforms.
- Textual support through the OpenModelica API and graphical support in OMEdit for generation of single or multiple requirement verification scenarios
- VVDRlib – a small library for connecting requirements and models together, with notions for mediators, scenarios, design alternatives

- Further enhanced OMPython including linearization, now also working with Python 3. ¨
- Jupyter notebooks also supported with OMPython and Python 3
- New enhanced library testing script (libraries.openmodelica.org/branches).
- Addition of mutable reference data types in MetaModelica
- Support for RedHat/Fedora binary builds of OpenModelica
- Support for exporting the system of equations in GraphML (yEd) format for debugging

-,col=changelog,group=component,format=table)

29.7 Release Notes for OpenModelica 1.11.0

- Dramatically improved compilation speed and performance, in particular for large models.
- 3D animation visualization of regular MSL MultiBody simulations and for real-time FMUs.
- Better support for synchronous and state machine language elements, now supports 90% of the clocked synchronous library.
- Several OMEdit improvements including folding of large annotations.
- 64-bit OM on Windows further stabilized
- An updated OMDev (OpenModelica Development Environment), involving msys2. This was needed for the shift to 64-bit on Windows.
- Integration of Sundials/IDA DAE solver with potentially large increase of simulation performance for large models with sparse structure.
- Improved library coverage.
- Parameter sensitivity analysis added to OMC.

29.7.1 OpenModelica Compiler (OMC)

- Real-time synchronization support by using simFlag -rt=1.0 (or some other time scaling factor).
- A prototype implementation of OPC UA using an open source OPC UA implementation. The old OPC implementation was not maintained and relied on a Windows-only proprietary OPC DA+UA package. (At the moment, OPC is experimental and lacks documentation; it only handles reading/writing Real/Boolean input/state variables. It is planned for OMEdit to use OPC UA to re-implement interactive simulations and plotting.)
- Dramatically improved compilation speed and dramatically reduced memory requirements for very large models. In Nov 2015, the largest power generation and transmission system model that OMC could handle had 60000 equations and it took 700 seconds to generate the simulation executable code; it now takes only 45 seconds to do so with OMC 1.11.0, which can also handle a model 10 times bigger (600 000 equations) in

less than 15 minutes and with less than 32 GB of RAM. Simulation times are comparable to domain-specific simulation tools. See for example ScalableTestSuite for some of the improvements.

- Improved library coverage
- Better support for synchronous and state machine language elements, now simulates 90% of the clocked synchronous library.
- Enhanced Cpp runtime to support the PowerSystems library.
- Integration of Sundials/IDA solver as an alternative to DASSL.
- A DAEMode solver mode was added, which allows to use the sparse IDA solver to handle the DAEs directly. This can lead to substantially faster simulation on large systems with sparse structure, compared to the traditional approach.
- The direct sparse solvers KLU and SuperLU have been added, with benefits for models with large algebraic loops.
- Multi-parameter sensitivity analysis added to OMC.
- Progress on more efficient inline function mechanism.
- Stabilized 64-bit Windows support.
- Performance improvement of parameter evaluation.
- Enhanced tearing support, with prefer iteration variables and user-defined tearing.
- Support for external object aliases in connectors and equations (a non-standard Modelica extension).
- Code generation directly to file (saves maximum memory used). #3356
- Code generation in parallel is enabled since #3356 (controlled by omc flag `-n`). This improves performance since generating code directly to file avoid memory allocation.
- Allowing mixed dense and sparse linear solvers in the generated simulation (chosen depending on simflags `-ls` (dense solver), `-lss` (sparse solver), `-lssMaxDensity` and `-lssMinSize`).

29.7.2 Graphic Editor OMEdit

- Significantly faster browsing of most libraries.
- Several GUI improvements including folding of multi-line annotations.
- Further improved code formatting preservation during edits.
- Support for all simulation logging flags.
- Select and export variables after simulation.
- Support for Byte Order Mark. Added support enables other tools to correctly read the files written by OMEdit.
- Save files with line endings according to OS (Windows (CRLF), Unix (LF)).

- Added OMEdit support for FMU cross compilation. This makes it possible to launch OMEdit on a remote or virtual Linux machine using a Windows X server and export an FMU with Windows binaries.
- Support of DisplayUnit and unit conversion.
- Fixed automatic save.
- Initial support for DynamicSelect in model diagrams (texts and visible attribute after simulation, no expressions yet).
- An HTML documentation editor (not WYSIWYG; that editor will be available in the subsequent release).
- Improved logging in OMEdit of structured messages and standard output streams for simulations.

29.7.3 FMI Support

- Cross compilation of C++ FMU export. Compared to the C runtime, the C++ cross compilation covers the whole runtime for model exchange.
- Improved Newton solver for C++ FMUs (scaling and step size control).

29.7.4 Other things

- 3D animation visualization of regular MSL MultiBody simulations and for real-time FMUs.
- An updated OMDev (OpenModelica Development Environment), involving msys2. This was needed for the shift to 64-bit on Windows.
- OMWebbook, a web version of OMNotebook online. Also, a script is available to convert an OMNotebook to an OMWebbook.
- A Jupyter notebook Modelica mode, available in OpenModelica.

1.11.0,status=closed,severity!=trivial,resolution=fixedl-,col=changelog,group=component,format=table)

29.8 Release Notes for OpenModelica 1.10.0

The most important enhancements in the OpenModelica 1.10.0 release:

29.8.1 OpenModelica Compiler (OMC)

New features:

- Real-time synchronization support by using simFlag -rt=1.0 (or some

other time scaling factor). - A prototype implementation of OPC UA using an open source OPC UA implementation. The old OPC implementation was not maintained and relied on a Windows-only proprietary OPC DA+UA package. (At the moment, OPC is experimental and lacks documentation; it only handles reading/writing Real/Boolean input/state variables. It is planned for OMEdit to use OPC UA to re-implement interactive simulations and plotting.)

Performance enhancements:

- Code generation directly to file (saves maximum memory used). #3356 -

Code generation in parallel enabled since #3356 allows this without allocating too much memory (controlled by omc flag `-n`). - Various scalability enhancements, allowing the compiler to handle hundreds of thousands of equations. See for example ScalableTestSuite for some of the improvements. - Better defaults for handling tearing (OMC flags `--maxSizeLinearTearing` and `--maxSizeNonlinearTearing`). - Allowing mixed dense and sparse linear solvers in the generated simulation (chosen depending on simflags `-ls` (dense solver), `-lss` (sparse solver), `-lssMaxDensity` and `-lssMinSize`).

29.8.2 Graphic Editor OMEdit

29.8.3 OpenModelica Notebook (OMNotebook)

29.8.4 Optimization

29.8.5 FMI Support

29.8.6 OpenModelica Development Environment (OMDev)

29.9 Release Notes for OpenModelica 1.9.4

OpenModelica v1.9.4 was released 2016-03-09. These notes cover the v1.9.4 release and its subsequent bug-fix releases (now up to 1.9.7).

29.9.1 OpenModelica Compiler (OMC)

- Improved simulation speed for many models. simulation speed went up for 80% of the models. The compiler frontend became faster for almost all models, average about 40% faster.
- Initial support for synchronous models with clocked equations as defined in the Modelica 3.3 standard
- Support for homotopy operator

29.9.2 Graphic Editor OMEdit

- Undo/Redo support.
- Preserving text formatting, including indentation and whitespace. This is especially important for diff/merge with several collaborating developers possibly using several different Modelica tools.
- Better support for inherited classes.
- Allow simulating models using visual studio compiler.
- Support for saving Modelica package in a folder structure.
- Allow reordering of classes inside a package.
- Highlight matching parentheses in text view.
- When copying the text retain the text highlighting and formatting.
- Support for global head definition in the documentation by using `__OpenModelica_infoHeader` annotation.
- Support for expandable connectors.
- Support for uses annotation.

29.9.3 FMI Support

- Full FMI 2.0 co-simulation support now available
- Upgrade Cpp runtime from C++03 to C++11 standard, minimizing external link dependencies. Exported FMUs don't depend on additional libraries such as boost anymore
- FMI 2.0 is broken for some models in 1.9.4. Upgrading to 1.9.6 is advised.

29.10 Release Notes for OpenModelica 1.9.3

The most important enhancements in the OpenModelica 1.9.3 release:

- Enhanced collaborative development and testing of OpenModelica by moving to the GIT-hub framework for versioning and parallel development.
- More accessible and up-to-date automatically generated documentation provided in both html and pdf.
- Further improved simulation speed and coverage of several libraries.
- OMEdit graphic connection editor improvements.
- OMNotebook improvements.

29.10.1 OpenModelica Compiler (OMC)

This release mainly includes improvements of the OpenModelica Compiler (OMC), including, but not restricted to the following:

- Further improved simulation speed and coverage for several libraries.
- Faster generated code for functions involving arrays, factor 2 speedup for many power generation models.
- Better initialization.
- An implicit inline Euler solver available.
- Code generation to enable vectorization of for-loops.
- Improved non-linear, linear and mixed system solving.
- Cross-compilation for the ARMhf architecture.
- A prototype state machine implementation.
- Improved performance and stability of the C++ runtime option.
- More accessible and up-to-date automatically generated documentation provided in both html and .pdf.

29.10.2 Graphic Editor OMEdit

There are several improvements to the OpenModelica graphic connection editor OMEdit:

- Support for uses annotations.
- Support for declaring components as vectors.
- Faster messages browser with clickable error messages.
- Support for managing the stacking order of graphical shapes.
- Several improvements to the plot tool and text editor in OMEdit.

29.10.3 OpenModelica Notebook (OMNotebook)

Several improvements:

- Support for moving cells from one place to another in a notebook.
- A button for evaluation of whole notebooks.
- A new cell type called Latex cells, supporting Latex formatted input that provides mathematical typesetting of formulae when evaluated.

29.10.4 Optimization

Several improvements of the Dynamic Optimization module with collocation, using Ipopt:

- Better performance due to smart treatment of algebraic loops for optimization.
- Improved formulation of optimization problems with an annotation approach which also allows graphical problem formulation.
- Proper handling of constraints at final time.

29.10.5 FMI Support

Further improved FMI 2.0 co-simulation support.

29.10.6 OpenModelica Development Environment (OMDev)

A big change: version handling and parallel development has been improved by moving from SVN to GIThub. This makes it easier for each developer to test his/her fixes and enhancements before committing the code. Automatic mirroring of all code is still performed to the OpenModelica SVN site.

29.11 Release Notes for OpenModelica 1.9.2

The OpenModelica 1.9.2 Beta release is available now, January 31, 2015. Please try it and give feedback! The final release is planned within 1-2 weeks after some more testing. The most important enhancements in the OpenModelica 1.9.2 release:

- The OpenModelica compiler has moved to a new development and release platform: the bootstrapped OpenModelica compiler. This gives advantages in terms of better programmability, maintenance, debugging, modularity and current/future performance increases.
- The OpenModelica graphic connection editor OMEdit has become 3-5 times faster due to faster communication with the OpenModelica compiler linked as a DLL. This was made possible by moving to the bootstrapped compiler.
- Further improved simulation coverage for a number of libraries.
- OMEdit graphic connection editor improvements

29.11.1 OpenModelica Compiler (OMC)

This release mainly includes improvements of the OpenModelica Compiler (OMC), including, but not restricted to the following:

- The OpenModelica compiler has moved to a new development and release platform: the bootstrapped OpenModelica compiler. This gives advantages in terms of better programmability, maintenance, debugging, modularity and current/future performance increases.
- Further improved simulation coverage for a number of libraries compared to 1.9.1. For example:
  - MSL 3.2.1 100% compilation, 97% simulation (3% increase)
  - MSL Trunk 99% compilation (1% increase), 93% simulation (3% increase)
  - ModelicaTest 3.2.1 99% compilation (2% increase), 95% simulation (6% increase)
  - ThermoSysPro 100% compilation, 80% simulation (17% increase)
  - ThermoPower 97% compilation (5% increase), 85% simulation (5% increase)
  - Buildings 80% compilation (1% increase), 73% simulation (1% increase)
- Further enhanced OMC compiler front-end coverage, scalability, speed and memory.
- Better initialization.
- Improved tearing.
- Improved non-linear, linear and mixed system solving.
- Common subexpression elimination support - drastically increases performance of some models.

29.11.2 Graphic Editor OMEdit

- The OpenModelica graphic connection editor OMEdit has become 3-5 times faster due to faster communication with the OpenModelica compiler linked as a DLL. This was made possible by moving to the bootstrapped compiler.
- Enhanced simulation setup window in OMEdit, which among other things include better support for integration methods and dassl options.
- Support for running multiple simultaneous simulation.
- Improved handling of modifiers.
- Re-simulate with changed options, including history support and re-simulating with previous options possibly edited.
- More user friendly user interface by improved connection line drawing, added snap to grid for icons and conversion of icons from PNG to SVG, and some additional fixes.

29.11.3 Optimization

Some smaller improvements of the Dynamic Optimization module with collocation, using Ipopt.

29.11.4 FMI Support

Further improved for FMI 2.0 model exchange import and export, now compliant according to the FMI compliance tests. FMI 1.0 support has been further improved.

29.12 Release Notes for OpenModelica 1.9.1

The most important enhancements in the OpenModelica 1.9.1 release:

- Improved library support.
- Further enhanced OMC compiler front-end coverage and scalability
- Significant improved simulation support for libraries using Fluid and Media.
- Dynamic model debugger for equation-based models integrated with OMEdit.
- Dynamic algorithm model debugger with OMEdit; including support for MetaModelica when using the bootstrapped compiler.

New features: Dynamic debugger for equation-based models; Dynamic Optimization with collocation built into OpenModelica, performance analyzer integrated with the equation model debugger.

29.12.1 OpenModelica Compiler (OMC)

This release mainly includes improvements of the OpenModelica Compiler (OMC), including, but not restricted to the following:

- Further improved OMC model compiler support for a number of libraries including MSL 3.2.1, ModelicaTest 3.2.1, PetriNet, Buildings, PowerSystems, OpenHydraulics, ThermoPower, and ThermoSysPro.
- Further enhanced OMC compiler front-end coverage, scalability, speed and memory.
- Better coverage of Modelica libraries using Fluid and Media.
- Automatic differentiation of algorithms and functions.
- Improved testing facilities and library coverage reporting.
- Improved model compilation speed by compiling model parts in parallel (bootstrapped compiler).
- Support for running model simulations in a web browser.
- New faster initialization that handles over-determined systems, under-determined systems, or both.
- Compiler back-end partly redesigned for improved scalability and better modularity.

- Better tearing support.
- The first run-time Modelica equation-based model debugger, not available in any other Modelica tool, integrated with OMEdit.
- Enhanced performance profiler integrated with the debugger.
- Improved parallelization prototype with several parallelization strategies, task merging and duplication, shorter critical paths, several scheduling strategies.
- Some support for general solving of mixed systems of equations.
- Better error messages.
- Improved bootstrapped OpenModelica compiler.
- Better handling of array subscripts and dimensions.
- Improved support for reduction functions and operators.
- Better support for partial functions.
- Better support for function tail recursion, which reduces memory usage.
- Partial function evaluation in the back-end to improve solving singular systems.
- Better handling of events/zero crossings.
- Support for colored Jacobians.
- New differentiation package that can handle a much larger number of expressions.
- Support for sparse solvers.
- Better handling of asserts.
- Improved array and matrix support.
- Improved overloaded operators support.
- Improved handling of overconstrained connection graphs.
- Better support for the cardinality operator.
- Parallel compilation of generated code for speeding up compilation.
- Split of model files into several for better compilation scalability.
- Default linear tearing.
- Support for impure functions.
- Better compilation flag documentation.
- Better automatic generation of documentation.
- Better support for calling functions via instance.
- New text template based unparsing for DAE, Absyn, SCode, TaskGraphs, etc.
- Better support for external objects (#2724, reject non-constructor functions returning external objects)

- Improved C++ runtime.
- Improved testing facilities.
- New unit checking implementation.
- Support for model rewriting expressions via rewriting rules in an external file.
- Reject more bad code (r19986, consider records with different components type-incompatible)

29.12.2 OpenModelica Connection Editor (OMEdit)

- Convenient editing of model parameter values and re-simulation without recompilation after parameter changes.
- Improved plotting.
- Better handling of flags/units/resources/crashes.
- Run-time Modelica equation-based model debugger that provides both dynamic run-time debugging and debugging of symbolic transformations.
- Run-time Modelica algorithmic code debugger; also MetaModelica debugger with the bootstrapped OpenModelica compiler.

29.12.3 OMPython

The interface was changed to version 2.0, which uses one object for each OpenModelica instance you want active. It also features a new and improved parser that returns easier to use datatypes like maps and lists.

29.12.4 Optimization

A builtin integrated Dynamic Optimization module with collocation, using Ipopt, is now available.

29.12.5 FMI Support

Support for FMI 2.0 model exchange import and export has been added. FMI 1.0 support has been further improved.

29.13 Release Notes for OpenModelica 1.9.0

This is the summary description of changes to OpenModelica from 1.8.1 to 1.9.0, released 2013-10-09. This release mainly includes improvements of the OpenModelica Compiler (OMC), including, but not restricted to the following:

29.13.1 OpenModelica Compiler (OMC)

This release mainly includes bug fixes and improvements of the OpenModelica Compiler (OMC), including, but not restricted to the following:

- A more stable and complete OMC model compiler. The 1.9.0 final version simulates many more models than the previous 1.8.1 version and OpenModelica 1.9.0 beta versions.
- Much better simulation support for MSL 3.2.1, now 270 out of 274 example models compile (98%) and 245 (89%) simulate, compared to 30% simulating in the 1.9.0 beta1 release.
- Much better simulation for the ModelicaTest 3.2.1 library, now 401 out of 428 models build (93%) and 364 simulate (85%), compared to 32% in November 2012.
- Better simulation support for several other libraries, e.g. more than twenty examples simulate from ThermoSysPro, and all but one model from PlanarMechanics simulate.
- Improved tearing algorithm for the compiler backend. Tearing is by default used.
- Much faster matching and dynamic state selection algorithms for the compiler backend.
- New index reduction algorithm implementation.
- New default initialization method that symbolically solves the initialization problem much faster and more accurately. This is the first version that in general initialize hybrid models correctly.
- Better class loading from files. The package.order file is now respected and the file structure is more thoroughly examined (#1764).
- It is now possible to translate the error messages in the omc kernel (#1767).
- FMI Support. FMI co-simulation with OpenModelica as master. Improved

FMI Import and export for model exchange. Most of FMI 2.0 is now also supported.

- Checking (when possible) that variables have been assigned to before they are used in algorithmic code (#1776).
- Full version of Python scripting.
- 3D graphics visualization using the Modelica3D library.
- The PySimulator package from DLR for additional analysis is integrated with OpenModelica (see Modelica2012 paper), and included in the OpenModelica distribution (Windows only).
- Prototype support for uncertainty computations, special feature enabled by special flag.

- Parallel algorithmic Modelica support (ParModelica) for efficient portable parallel algorithmic programming based on the OpenCL standard, for CPUs and GPUs.
- Support for optimization of semiLinear according to MSL 3.3 chapter 3.7.2.5 semiLinear (r12657,r12658).
- The compiler is now fully bootstrapped and can compile itself using a modest amount of heap and stack space (less than the RML-based compiler, which is still the default).
- Some old debug-flags were removed. Others were renamed. Debug flags can now be enabled by default.
- Removed old unused simulation flags noClean and storeInTemp (r15927).
- Many stack overflow issues were resolved.
- Dynamic Optimization with OpenModelica. Dynamic optimization with XML export to the CasADi package is now integrated with OpenModelica. Moreover, a native integrated Dynamic Optimization prototype using Ipopt is now in the OpenModelica release, but currently needs a special flag to be turned on since it needs more testing and refinement before being generally made available.

29.13.2 OpenModelica Notebook (OMNotebook)

- A `shortOutput` option has been introduced in the simulate command

for less verbose output. The DrModelica interactive document has been updated and the models tested. Almost all models now simulate with OpenModelica.

29.13.3 OpenModelica Eclipse Plug-in (MDT)

- Enhanced debugger for algorithmic Modelica code, supporting both

standard Modelica algorithmic code called from simulation models, and MetaModelica code.

29.13.4 OpenModelica Development Environment (OMDev)

- Migration of version handling and configuration management from

CodeBeamer to Trac.

29.13.5 Graphic Editor OMEdit

- General GUI: backward and forward navigation support in Documentation view, enhanced parameters window with support for Dialog annotation. Most of the images are converted from raster to vector graphics i.e PNG to SVG.
- Libraries Browser: better loading of libraries, library tree can now show protected classes, show library items class names as middle ellipses if the class name text is larger, more options via the right click menu for quick usage.

- ModelWidget: add the partial class as a replaceable component, look for the default component prefixes and name when adding the component.
- GraphicsView: coordinate system manipulation for icon and diagram layers. Show red box for models that do not exist. Show default graphical annotation for the components that doesn't have any graphical annotations. Better resizing of the components. Properties dialog for primitive shapes i.e Line, Polygon, Rectangle, Ellipse, Text and Bitmap.
- File Opening: open one or more Modelica files, allow users to select the encoding while opening the file, convert files to UTF-8 encoding, allow users to open the OpenModelica result files.
- Variables Browser: find variables in the variables browser, sorting in the variables browser.
- Plot Window: clear all curves of the plot window, preserve the old selected variable and update its value with the new simulation result.
- Simulation: support for all the simulation flags, read the simulation output as soon as is is obtained, output window for simulations, options to set matching algorithm and index reduction method for simulation. Display all the files generated during the simulation is now supported. Options to set OMC command line flags.
- Options: options for loading libraries via loadModel and loadFile each time GUI starts, save the last open file directory location, options for setting line wrap mode and syntax highlighting.
- Modelica Text Editor: preserving user customizations, new search & replace functionality, support for comment/uncomment.
- Notifications: show custom dialogs to users allowing them to choose whether they want to see this dialog again or not.
- Model Creation: Better support for creating new classes. Easy creation of extends classes or nested classes.
- Messages Widget: Multi line error messages are now supported.
- Crash Detection: The GUI now automatically detects the crash and writes a stack trace file. The user is given an option to send a crash report along with the stack trace file and few other useful files via email.
- Autosave: OMEdit saves the currently edited model regularly, in order to avoid losing edits after GUI or compiler crash. The save interval can be set in the Options menu.

29.13.6 ModelicaML

- Enhanced ModelicaML version with support for value bindings in

requirements-driven modeling available for the latest Eclipse and Papyrus versions. GUI specific adaptations. Automated model composition workflows (used for model-based design verification against requirements) are modularized and have improved in terms of performance.

29.14 Release Notes for OpenModelica 1.8.1

The OpenModelica 1.8.1 release has a faster and more stable OMC model compiler. It flattens and simulates more models than the previous 1.8.0 version. Significant flattening speedup of the compiler has been achieved for certain large models. It also contains a New ModelicaML version with support for value bindings in requirements-driven modeling and importing Modelica library models into ModelicaML models. A beta version of the new OpenModelica Python scripting is also included. The release was made on 2012-04-03 (r11645).

29.14.1 OpenModelica Compiler (OMC)

This release includes bug fixes and improvements of the flattening frontend part of the OpenModelica Compiler (OMC) and several improvements of the backend, including, but not restricted to:

- A faster and more stable OMC model compiler. The 1.8.1 version flattens and simulates more models than the previous 1.8.0 version.
- Support for operator overloading (except Complex numbers).
- New ModelicaML version with support for value bindings in requirements-driven modeling and importing Modelica library models into ModelicaML models.
- Faster plotting in OMNotebook. The feature sendData has been removed from OpenModelica. As a result, the kernel no longer depends on Qt. The plot3() family of functions have now replaced to plot(), which in turn have been removed. The non-standard visualize() command has been removed in favour of more recent alternatives.
- Store OpenModelica documentation as Modelica Documentation annotations.
- Re-implementation of the simulation runtime using C instead of C++ (this was needed to export FMI source-based packages).
- FMI import/export bug fixes.
- Changed the internal representation of various structures to share more memory. This significantly improved the performance for very large models that use records.
- Faster model flattening, Improved simulation, some graphical API bug fixes.
- More robust and general initialization, but currently time-consuming.
- New initialization flags to omc and options to simulate(), to control whether fast or robust initialization is selected, or initialization from an external (.mat) data file.
- New options to API calls list, loadFile, and more.
- Enforce the restriction that input arguments of functions may not be assigned to.
- Improved the scripting environment. cl := $TypeName(Modelica);getClassComment(cl); now works as expected. As does looping over lists of typenames and using reduction expressions.
- Beta version of Python scripting.
- Various bugfixes.

- NOTE: interactive simulation is not operational in this release. It will be put back again in the near future, first available as a nightly build. It is also available in the previous 1.8.0 release.

29.14.2 OpenModelica Notebook (OMNotebook)

- Faster and more stable plottning.

29.14.3 OpenModelica Shell (OMShell)

- No changes.

29.14.4 OpenModelica Eclipse Plug-in (MDT)

- Small fixes and improvements.

29.14.5 OpenModelica Development Environment (OMDev)

- No changes.

29.14.6 Graphic Editor OMEdit

- Bug fixes.

29.14.7 OMOptim Optimization Subsystem

- Bug fixes.

29.14.8 FMI Support

- Bug fixes.

29.15 OpenModelica 1.8.0, November 2011

The OpenModelica 1.8.0 release contains OMC flattening improvements for the Media library - it now flattens the whole library and simulates about 20% of its example models. Moreover, about half of the Fluid library models also flatten. This release also includes two new tool functionalities - the FMI for model exchange import and export, and a new efficient Eclipse-based debugger for Modelica/MetaModelica algorithmic code.

29.15.1 OpenModelica Compiler (OMC)

This release includes bug fixes and improvements of the flattening frontend part of the OpenModelica Compiler (OMC) and several improvements of the backend, including, but not restricted to: A faster and more stable OMC model compiler. The 1.8.0 version flattens and simulates more models than the previous 1.7.0 version.

- Flattening of the whole Media library, and about half of the Fluid

library. Simulation of approximately 20% of the Media library example models. - Functional Mockup Interface FMI 1.0 for model exchange, export and import, for the Windows platform. - Bug fixes in the OpenModelica graphical model connection editor OMEdit, supporting easy-to-use graphical drag-and-drop modeling and MSL 3.1. - Bug fixes in the OMOptim optimization subsystem. - Beta version of compiler support for a new Eclipse-based very efficient algorithmic code debugger for functions in MetaModelica/Modelica, available in the development environment when using the bootstrapped OpenModelica compiler. - Improvements in initialization of simulations. - Improved index reduction with dynamic state selection, which improves simulation. - Better error messages from several parts of the compiler, including a new API call for giving better error messages. - Automatic partitioning of equation systems and multi-core parallel simulation of independent parts based on the shared-memory OpenMP model. This version is a preliminary experimental version without load balancing.

29.15.2 OpenModelica Notebook (OMNotebook)

No changes.

29.15.3 OpenModelica Shell (OMShell)

Small performance improvements.

29.15.4 OpenModelica Eclipse Plug-in (MDT)

Small fixes and improvements. MDT now also includes a beta version of a new Eclipse-based very efficient algorithmic code debugger for functions in MetaModelica/Modelica.

29.15.5 OpenModelica Development Environment (OMDev)

Third party binaries, including Qt libraries and executable Qt clients, are now part of the OMDev package. Also, now uses GCC 4.4.0 instead of the earlier GCC 3.4.5.

29.15.6 Graphic Editor OMEdit

Bug fixes. Access to FMI Import/Export through a pull-down menu. Improved configuration of library loading. A function to go to a specific line number. A button to cancel an on-going simulation. Support for some updated OMC API calls.

29.15.7 New OMOptim Optimization Subsystem

Bug fixes, especially in the Linux version.

29.15.8 FMI Support

The Functional Mockup Interface FMI 1.0 for model exchange import and export is supported by this release. The functionality is accessible via API calls as well as via pull-down menu commands in OMEdit.

29.16 OpenModelica 1.7.0, April 2011

The OpenModelica 1.7.0 release contains OMC flattening improvements for the Media library, better and faster event handling and simulation, and fast MetaModelica support in the compiler, enabling it to compiler itself. This release also includes two interesting new tools – the OMOptim optimization subsystem, and a new performance profiler for equation-based Modelica models.

29.16.1 OpenModelica Compiler (OMC)

This release includes bug fixes and performance improvements of the flattening frontend part of the OpenModelica Compiler (OMC) and several improvements of the backend, including, but not restricted to:

- Flattening of the whole Modelica Standard Library 3.1 (MSL 3.1),

except Media and Fluid. - Progress in supporting the Media library, some models now flatten. - Much faster simulation of many models through more efficient handling of alias variables, binary output format, and faster event handling. - Faster and more stable simulation through new improved event handling, which is now default. - Simulation result storage in binary .mat files, and plotting from such files. - Support for Unicode characters in quoted Modelica identifiers, including Japanese and Chinese. - Preliminary MetaModelica 2.0 support. (use setCommandLineOptions({“+g=MetaModelica”})). Execution is as fast as MetaModelica 1.0, except for garbage collection. - Preliminary bootstrapped OpenModelica compiler: OMC now compiles itself, and the bootstrapped compiler passes the test suite. A garbage collector is still missing. - Many bug fixes.

29.16.2 OpenModelica Notebook (OMNotebook)

Improved much faster and more stable 2D plotting through the new OMPlot module. Plotting from binary .mat files. Better integration between OMEdit and OMNotebook, copy/paste between them.

29.16.3 OpenModelica Shell (OMShell)

Same as previously, except the improved 2D plotting through OMPlot.

29.16.4 Graphic Editor OMEdit

Several enhancements of OMEdit are included in this release. Support for Icon editing is now available. There is also an improved much faster 2D plotting through the new OMPlot module. Better integration between OMEdit and OMNotebook, with copy/paste between them. Interactive on-line simulation is available in an easy-to-use way.

29.16.5 New OMOptim Optimization Subsystem

A new optimization subsystem called OMOptim has been added to OpenModelica. Currently, parameter optimization using genetic algorithms is supported in this version 0.9. Pareto front optimization is also supported.

29.16.6 New Performance Profiler

A new, low overhead, performance profiler for Modelica models has been developed.

29.17 OpenModelica 1.6.0, November 2010

The OpenModelica 1.6.0 release primarily contains flattening, simulation, and performance improvements regarding Modelica Standard Library 3.1 support, but also has an interesting new tool – the OMEdit graphic connection editor, and a new educational material called DrControl, and an improved ModelicaML UML/Modelica profile with better support for modeling and requirement handling.

29.17.1 OpenModelica Compiler (OMC)

This release includes bug fix and performance improvemetns of the flattening frontend part of the OpenModelica Compiler (OMC) and some improvements of the backend, including, but not restricted to:

- Flattening of the whole Modelica Standard Library 3.1 (MSL 3.1),

except Media and Fluid. - Improved flattening speed of a factor of 5-20 compared to OpenModelica 1.5 for a number of models, especially in the MultiBody library. - Reduced memory consumption by the OpenModelica compiler frontend, for certain large models a reduction of a factor 50. - Reorganized, more modular OpenModelica compiler backend, can now handle approximately 30 000 equations, compared to previously approximately 10 000 equations. - Better error messages from the compiler, especially regarding functions. - Improved simulation coverage of MSL 3.1. Many models that did not simulate before are now simulating. However, there are still many models in certain sublibraries that do not simulate. - Progress in supporting the Media library, but simulation is not yet possible. - Improved support for enumerations, both in the frontend and the backend. - Implementation of stream connectors. - Support for linearization through symbolic Jacobians. - Many bug fixes.

29.17.2 OpenModelica Notebook (OMNotebook)

A new DrControl electronic notebook for teaching control and modeling with Modelica.

29.17.3 OpenModelica Development Environment (OMDev)

Several enhancements. Support for match-expressions in addition to matchcontinue. Support for real if-then-else. Support for if-then without else-branches. Modelica Development Tooling 0.7.7 with small improvements such as more settings, improved error detection in console, etc.

29.17.4 New Graphic Editor OMEdit

A new improved open source graphic model connection editor called OMEdit, supporting 3.1 graphical annotations, which makes it possible to move models back and forth to other tools without problems. The editor has been implemented by students at Linköping University and is based on the C++ Qt library.

29.18 OpenModelica 1.5.0, July 2010

This OpenModelica 1.5 release has major improvements in the OpenModelica compiler frontend and some in the backend. A major improvement of this release is full flattening support for the MultiBody library as well as limited simulation support for MultiBody. Interesting new facilities are the interactive simulation and the integrated UML-Modelica modeling with ModelicaML. Approximately 4 person-years of additional effort have been invested in the compiler compared to the 1.4.5 version, e.g., in order to have a more complete coverage of Modelica 3.0, mainly focusing on improved flattening in the compiler frontend.

29.18.1 OpenModelica Compiler (OMC)

This release includes major improvements of the flattening frontend part of the OpenModelica Compiler (OMC) and some improvements of the backend, including, but not restricted to:

- Improved flattening speed of at least a factor of 10 or more compared

to the 1.4.5 release, primarily for larger models with inner-outer, but also speedup for other models, e.g. the robot model flattens in approximately 2 seconds. - Flattening of all MultiBody models, including all elementary models, breaking connection graphs, world object, etc. Moreover, simulation is now possible for at least five MultiBody models: Pendulum, DoublePendulum, InitSpringConstant, World, PointGravityWithPointMasses. - Progress in supporting the Media library, but simulation is not yet possible. - Support for enumerations, both in the frontend and the backend. - Support for expandable connectors. - Support for the inline and late inline annotations in functions. - Complete support for record constructors, also for records containing other records. - Full support for iterators, including nested ones. - Support for inferred iterator and for-loop ranges. - Support for the function derivative annotation. - Prototype of interactive simulation. - Prototype of integrated UML-Modelica modeling and simulation with ModelicaML. - A new bidirectional external Java interface for calling external Java functions, or for calling Modelica functions from Java. - Complete implementation of replaceable model extends. - Fixed problems involving arrays of unknown dimensions. - Limited support for tearing. - Improved error handling at division by zero. - Support for Modelica 3.1 annotations. - Support for all MetaModelica language constructs inside OpenModelica. - OpenModelica works also under 64-bit Linux and Mac 64-bit OSX. - Parallel builds and running test suites in parallel on multi-core platforms. - New OpenModelica text template language for easier implementation of code generators, XML generators, etc. - New OpenModelica code generators to C and C# using the text template language. - Faster simulation result data file output optionally as comma-separated values. - Many bug fixes.

It is now possible to graphically edit models using parts from the Modelica Standard Library 3.1, since the simForge graphical editor (from Politecnico di Milano) that is used together with OpenModelica has been updated to version 0.9.0 with a important new functionality, including support for Modelica 3.1 and 3.0 annotations. The 1.6 and 2.2.1 Modelica graphical annotation versions are still supported.

29.18.2 OpenModelica Notebook (OMNotebook)

Improvements in platform availability.

- Support for 64-bit Linux. - Support for Windows 7. - Better support

for MacOS, including 64-bit OSX.

29.19 OpenModelica 1.4.5, January 2009

This release has several improvements, especially platform availability, less compiler memory usage, and supporting more aspects of Modelica 3.0.

29.19.1 OpenModelica Compiler (OMC)

This release includes small improvements and some bugfixes of the OpenModelica Compiler (OMC):

- Less memory consumption and better memory management over time. This

also includes a better API supporting automatic memory management when calling C functions from within the compiler. - Modelica 3.0 parsing support. - Export of DAE to XML and MATLAB. - Support for several platforms Linux, MacOS, Windows (2000, Xp, Vista). - Support for record and strings as function arguments. - Many bug fixes. - (Not part of OMC): Additional free graphic editor SimForge can be used with OpenModelica.

29.19.2 OpenModelica Notebook (OMNotebook)

A number of improvements, primarily in the plotting functionality and platform availability.

- A number of improvements in the plotting functionality: scalable

plots, zooming, logarithmic plots, grids, etc. - Programmable plotting accessible through a Modelica API. - Simple 3D visualization. - Support for several platforms Linux, MacOS, Windows (2000, Xp, Vista).

29.20 OpenModelica 1.4.4, Feb 2008

This release is primarily a bug fix release, except for a preliminary version of new plotting functionality available both from the OMNotebook and separately through a Modelica API. This is also the first release under the open source license OSMC-PL (Open Source Modelica Consortium Public License), with support from the recently created Open Source Modelica Consortium. An integrated version handler, bug-, and issue tracker has also been added.

29.20.1 OpenModelica Compiler (OMC)

This release includes small improvements and some bugfixes of the OpenModelica Compiler (OMC):

- Better support for if-equations, also inside when. - Better support

for calling functions in parameter expressions and interactively through dynamic loading of functions. - Less memory consumtion during compilation and interactive evaluation. - A number of bug-fixes.

29.20.2 OpenModelica Notebook (OMNotebook)

Test release of improvements, primarily in the plotting functionality and platform availability.

- Preliminary version of improvements in the plotting functionality:

scalable plots, zooming, logarithmic plots, grids, etc., currently available in a preliminary version through the plot2 function. - Programmable plotting accessible through a Modelica API.

29.20.3 OpenModelica Eclipse Plug-in (MDT)

This release includes minor bugfixes of MDT and the associated MetaModelica debugger.

29.20.4 OpenModelica Development Environment (OMDev)

Extended test suite with a better structure. Version handling, bug tracking, issue tracking, etc. now available under the integrated Codebeamer.

29.21 OpenModelica 1.4.3, June 2007

This release has a number of significant improvements of the OMC compiler, OMNotebook, the MDT plugin and the OMDev. Increased platform availability now also for Linux and Macintosh, in addition to Windows. OMShell is the same as previously, but now ported to Linux and Mac.

29.21.1 OpenModelica Compiler (OMC)

This release includes a number of improvements of the OpenModelica Compiler (OMC):

- Significantly increased compilation speed, especially with large

models and many packages. - Now available also for Linux and Macintosh platforms. - Support for when-equations in algorithm sections, including elsewhen. - Support for inner/outer prefixes of components (but without type error checking). - Improved solution of nonlinear systems. - Added ability to compile generated simulation code using Visual Studio compiler. - Added "smart setting of fixed attribute to false. If initial equations, OMC instead has fixed=true as default for states due to allowing overdetermined initial equation systems. - Better state select heuristics. - New function getIncidenceMatrix(ClassName) for dumping the incidence matrix. - Builtin functions String(), product(), ndims(), implemented. - Support for terminate() and assert() in equations. - In emitted flat form: protected variables are now prefixed with protected when printing flat class. - Some support for tables, using omcTableTimeIni instead of dymTableTimeIni2. - Better support for empty arrays, and support for matrix operations like a*[1,2;3,4]. - Improved val() function can now evaluate array elements and record fields, e.g. val(x[n]), val(x.y) . - Support for reinit in algorithm sections. - String support in external functions. - Double precision floating point precision now also for interpreted expressions - Better simulation error messages. - Support for der(expressions). - Support for iterator expressions such as {3*i for i in 1..10}. - More test cases in the test suite. - A number of bug fixes, including sample and event handling bugs.

29.21.2 OpenModelica Notebook (OMNotebook)

A number of improvements, primarily in the platform availability.

- Available on the Linux and Macintosh platforms, in addition to

Windows. - Fixed cell copying bugs, plotting of derivatives now works, etc.

29.21.3 OpenModelica Shell (OMShell)

Now available also on the Macintosh platform.

29.21.4 OpenModelica Eclipse Plug-in (MDT)

This release includes major improvements of MDT and the associated MetaModelica debugger:

- Greatly improved browsing and code completion works both for standard

Modelica and for MetaModelica. - Hovering over identifiers displays type information. - A new and greatly improved implementation of the debugger for MetaModelica algorithmic code, operational in Eclipse. Greatly improved performance - only approx 10% speed reduction even for 100 000 line programs. Greatly improved single stepping, step over, data structure browsing, etc. - Many bug fixes.

29.21.5 OpenModelica Development Environment (OMDev)

Increased compilation speed for MetaModelica. Better if-expression support in MetaModelica.

29.22 OpenModelica 1.4.2, October 2006

This release has improvements and bug fixes of the OMC compiler, OMNotebook, the MDT plugin and the OMDev. OMShell is the same as previously.

29.22.1 OpenModelica Compiler (OMC)

This release includes further improvements of the OpenModelica Compiler (OMC):

- Improved initialization and index reduction. - Support for integer

arrays is now largely implemented. - The val(variable,time) scripting function for accessing the value of a simulation result variable at a certain point in the simulated time. - Interactive evalution of for-loops, while-loops, if-statements, if-expressions, in the interactive scripting mode. - Improved documentation and examples of calling the Model Query and Manipulation API. - Many bug fixes.

29.22.2 OpenModelica Notebook (OMNotebook)

Search and replace functions have been added. The DrModelica tutorial (all files) has been updated, obsolete sections removed, and models which are not supported by the current implementation marked clearly. Automatic recognition of the .onb suffix (e.g. when double-clicking) in Windows makes it even more convenient to use.

29.22.3 OpenModelica Eclipse Plug-in (MDT)

Two major improvements are added in this release:

- Browsing and code completion works both for standard Modelica and for

MetaModelica. - The debugger for algorithmic code is now available and operational in Eclipse for debugging of MetaModelica programs.

29.23 OpenModelica 1.4.1, June 2006

This release has only improvements and bug fixes of the OMC compiler, the MDT plugin and the OMDev components. The OMShell and OMNotebook are the same.

29.23.1 OpenModelica Compiler (OMC)

This release includes further improvements of the OpenModelica Compiler (OMC):

- Support for external objects. - OMC now reports the version number

(via command line switches or CORBA API getVersion()). - Implemented caching for faster instantiation of large models. - Many bug fixes.

29.23.2 OpenModelica Eclipse Plug-in (MDT)

Improvements of the error reporting when building the OMC compiler. The errors are now added to the problems view. The latest MDT release is version 0.6.6 (2006-06-06).

29.23.3 OpenModelica Development Environment (OMDev)

Small fixes in the MetaModelica compiler. MetaModelica Users Guide is now part of the OMDev release. The latest OMDev was release in 2006-06-06.

29.24 OpenModelica 1.4.0, May 2006

This release has a number of improvements described below. The most significant change is probably that OMC has now been translated to an extended subset of Modelica (MetaModelica), and that all development of the compiler is now done in this version..

29.24.1 OpenModelica Compiler (OMC)

This release includes further improvements of the OpenModelica Compiler (OMC):

- Partial support for mixed system of equations. - New initialization

routine, based on optimization (minimizing residuals of initial equations). - Symbolic simplification of builtin operators for vectors and matrices. - Improved code generation in simulation code to support e.g. Modelica functions. - Support for classes extending basic types, e.g. connectors (support for MSL 2.2 block connectors). - Support for parametric plotting via the plotParametric command. - Many bug fixes.

29.24.2 OpenModelica Shell (OMShell)

Essentially the same OMShell as in 1.3.1. One difference is that now all error messages are sent to the command window instead of to a separate log window.

29.24.3 OpenModelica Notebook (OMNotebook)

Many significant improvements and bug fixes. This version supports graphic plots within the cells in the notebook. Improved cell handling and Modelica code syntax highlighting. Command completion of the most common OMC commands is now supported. The notebook has been used in several courses.

29.24.4 OpenModelica Eclipse Plug-in (MDT)

This is the first really useful version of MDT. Full browsing of Modelica code, e.g. the MSL 2.2, is now supported. (MetaModelica browsing is not yet fully supported). Full support for automatic indentation of Modelica code, including the MetaModelica extensions. Many bug fixes. The Eclipse plug-in is now in use for OpenModelica development at PELAB and MathCore Engineering AB since approximately one month.

29.24.5 OpenModelica Development Environment (OMDev)

The following mechanisms have been put in place to support OpenModelica development.

- A separate web page for OMDev (OpenModelica Development Environment).
- A pre-packaged OMDev zip-file with precompiled binaries for

development under Windows using the mingw Gnu compiler from the Eclipse MDT plug-in. (Development is also possible using Visual Studio). - All source code of the OpenModelica compiler has recently been translated to an extended subset of Modelica, currently called MetaModelica. The current size of OMC is approximately 100 000 lines All development is now done in this version. - A new tutorial and users guide for development in MetaModelica. - Successful builds and tests of OMC under Linux and Solaris.

29.25 OpenModelica 1.3.1, November 2005

This release has several important highlights.

This is also the *first* release for which the New BSD (Berkeley) open-source license applies to the source code, including the whole compiler and run-time system. This makes is possible to use OpenModelica for both academic and commercial purposes without restrictions.

29.25.1 OpenModelica Compiler (OMC)

This release includes a significantly improved OpenModelica Compiler (OMC):

- Support for hybrid and discrete-event simulation (if-equations,

if-expressions, when-equations; not yet if-statements and when-statements). - Parsing of full Modelica 2.2 - Improved support for external functions. - Vectorization of function arguments; each-modifiers, better implementation of replaceable, better handling of structural parameters, better support for vector and array operations, and many other improvements. - Flattening of the Modelica Block library version 1.5 (except a few models), and simulation of most of these. - Automatic index reduction (present also in previous release). - Updated User's Guide including examples of hybrid simulation and external functions.

29.25.2 OpenModelica Shell (OMShell)

An improved window-based interactive command shell, now including command completion and better editing and font size support.

29.25.3 OpenModelica Notebook (OMNotebook)

A free implementation of an OpenModelica notebook (OMNotebook), for electronic books with course material, including the DrModelica interactive course material. It is possible to simulate and plot from this notebook.

29.25.4 OpenModelica Eclipse Plug-in (MDT)

An early alpha version of the first Eclipse plug-in (called MDT for Modelica Development Tooling) for Modelica Development. This version gives compilation support and partial support for browsing Modelica package hierarchies and classes.

29.25.5 OpenModelica Development Environment (OMDev)

The following mechanisms have been put in place to support OpenModelica development.

- Bugzilla support for OpenModelica bug tracking, accessible to anybody.
- A system for automatic regression testing of the compiler and

simulator, (+ other system parts) usually run at check in time. - Version handling is done using SVN, which is better than the previously used CVS system. For example, name change of modules is now possible within the version handling system.

第30章　Contributors to OpenModelica

This Appendix lists the individuals who have made significant contributions to OpenModelica, in the form of software development, design, documentation, project leadership, tutorial material, promotion, etc. The individuals are listed for each year, from 1998 to the current year: the project leader and main author/editor of this document followed by main contributors followed by contributors in alphabetical order.

30.1 OpenModelica Contributors 2015

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Volker Waurich, TU Dresden, Dresden, Germany.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Anders Andersson, VTI, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Robert Braun, IEI, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Daniel Bouskela, EDF, Paris, France.

Lena Buffoni, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy.

Atiyah Elsheikh, AIT, Vinnea, Austria.

Rüdiger Franke, ABB, Germany

Jens Frenkel, TU Dresden, Dresden, Germany.

Mahder Gebremedhin, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.

Henning Kiel, Bocholt, Germany.

Tommi Karhela, VTT, Espoo, Finland.

Petter Krus, IEI, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland.

Leonardo Laguna, Wolfram MathCore AB, Linköping, Sweden.

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.

Oliver Lenord, Siemens PLM, California, USA.

Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.

Alachew Mengist, PELAB, Linköping University, Linköping, Sweden.

Abhir Raj Metkar, CDAC, Trivandrum, Kerala, India.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Lars Mikelsons, Bosch Rexroth, Lohr am Main, Germany.

Afshin Moghadam, PELAB, Linköping University, Linköping, Sweden.

Kannan Moudgalya, IIT Bombay, Mumbai, India.

Kenneth Nealy, USA.

Maroun Nemer, CEP Paristech, Ecole des Mines, Paris, France.

Hannu Niemistö, VTT, Espoo, Finland.

Peter Nordin, IEI, Linköping University, Linköping, Sweden.

Arunkumar Palanisamy, PELAB, Linköping University, Linköping, Sweden.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Vitalij Ruge, Fachhochschule Bielefeld, Bielefeld, Germany.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Roland Samlaus, Bosch, Stuttgart, Germany.

Wladimir Schamai, EADS, Hamburg, Germany.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany.

Jan Šilar, Charles University, Prague, Czech Republic

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.

Bernhard Thiele, PELAB, Linköping University, Linköping, Sweden

Hubert Thierot, CEP Paristech, Ecole des Mines, Paris, France.

Gustaf Thorslund, PELAB, Linköping University, Linköping, Sweden.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Marcus Walther, TU Dresden, Dresden, Germany

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.

30.2 OpenModelica Contributors 2014

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

Jens Frenkel, TU Dresden, Dresden, Germany.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Robert Braun, IEI, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Stefan Brus, PELAB, Linköping University, Linköping, Sweden.

Lena Buffoni, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy.

Filippo Donida, Politecnico di Milano, Milan, Italy.

Mahder Gebremedhin, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Zoheb Hossain, PELAB, Linköping University, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.

Tommi Karhela, VTT, Espoo, Finland.

Petter Krus, IEI, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland.

Abhinn Kothari, PELAB, Linköping University, Linköping, Sweden.

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.

Oliver Lenord, Siemens PLM, California, USA.

Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.

Henrik Magnusson, Linköping, Sweden.

Abhi Raj Metkar, CDAC, Trivandrum, Kerala, India.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Tuomas Miettinen, VTT, Espoo, Finland.

Afshin Moghadam, PELAB, Linköping University, Linköping, Sweden.

Maroun Nemer, CEP Paristech, Ecole des Mines, Paris, France.

Hannu Niemistö, VTT, Espoo, Finland.

Peter Nordin, IEI, Linköping University, Linköping, Sweden.

Arunkumar Palanisamy, PELAB, Linköping University, Linköping, Sweden.

Karl Pettersson, IEI, Linköping University, Linköping, Sweden.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Jhansi Remala, PELAB, Linköping University, Linköping, Sweden.

Reino Ruusu, VTT, Espoo, Finland.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Wladimir Schamai, EADS, Hamburg, Germany.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany.

Alachew Shitahun, PELAB, Linköping University, Linköping, Sweden.

Anton Sodja, University of Ljubljana, Ljubljana, Slovenia

Ingo Staack, IEI, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.

Hubert Thierot, CEP Paristech, Ecole des Mines, Paris, France.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Parham Vasaiely, EADS, Hamburg, Germany.

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.

Robert Wotzlaw, Goettingen, Germany.

Azam Zia, PELAB, Linköping University, Linköping, Sweden.

30.3 OpenModelica Contributors 2013

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

Jens Frenkel, TU Dresden, Dresden, Germany.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Robert Braun, IEI, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Stefan Brus, PELAB, Linköping University, Linköping, Sweden.

Lena Buffoni, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy.

Filippo Donida, Politecnico di Milano, Milan, Italy.

Mahder Gebremedhin, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Zoheb Hossain, PELAB, Linköping University, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.

Tommi Karhela, VTT, Espoo, Finland.

Petter Krus, IEI, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland.

Abhinn Kothari, PELAB, Linköping University, Linköping, Sweden.

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.

Oliver Lenord, Siemens PLM, California, USA.

Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.

Henrik Magnusson, Linköping, Sweden.

Abhi Raj Metkar, CDAC, Trivandrum, Kerala, India.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Tuomas Miettinen, VTT, Espoo, Finland.

Afshin Moghadam, PELAB, Linköping University, Linköping, Sweden.

Maroun Nemer, CEP Paristech, Ecole des Mines, Paris, France.

Hannu Niemistö, VTT, Espoo, Finland.

Peter Nordin, IEI, Linköping University, Linköping, Sweden.

Arunkumar Palanisamy, PELAB, Linköping University, Linköping, Sweden.

Karl Pettersson, IEI, Linköping University, Linköping, Sweden.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Jhansi Remala, PELAB, Linköping University, Linköping, Sweden.

Reino Ruusu, VTT, Espoo, Finland.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Wladimir Schamai, EADS, Hamburg, Germany.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany.

Alachew Shitahun, PELAB, Linköping University, Linköping, Sweden.

Anton Sodja, University of Ljubljana, Ljubljana, Slovenia

Ingo Staack, IEI, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.

Hubert Thierot, CEP Paristech, Ecole des Mines, Paris, France.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Parham Vasaiely, EADS, Hamburg, Germany.

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.

Robert Wotzlaw, Goettingen, Germany.

Azam Zia, PELAB, Linköping University, Linköping, Sweden.

30.4 OpenModelica Contributors 2012

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

Jens Frenkel, TU Dresden, Dresden, Germany.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Mikael Axin, IEI, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Robert Braun, IEI, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Stefan Brus, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy.

Filippo Donida, Politecnico di Milano, Milan, Italy.

Mahder Gebremedhin, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Zoheb Hossain, PELAB, Linköping University, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.

Tommi Karhela, VTT, Espoo, Finland.

Petter Krus, IEI, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland.

Abhinn Kothari, PELAB, Linköping University, Linköping, Sweden.

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.

Oliver Lenord, Siemens PLM, California, USA.

Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.

Henrik Magnusson, Linköping, Sweden.

Abhi Raj Metkar, CDAC, Trivandrum, Kerala, India.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Tuomas Miettinen, VTT, Espoo, Finland.

Afshin Moghadam, PELAB, Linköping University, Linköping, Sweden.

Maroun Nemer, CEP Paristech, Ecole des Mines, Paris, France.

Hannu Niemistö, VTT, Espoo, Finland.

Peter Nordin, IEI, Linköping University, Linköping, Sweden.

Arunkumar Palanisamy, PELAB, Linköping University, Linköping, Sweden.

Karl Pettersson, IEI, Linköping University, Linköping, Sweden.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Jhansi Remala, PELAB, Linköping University, Linköping, Sweden.

Reino Ruusu, VTT, Espoo, Finland.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Wladimir Schamai, EADS, Hamburg, Germany.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany.

Alachew Shitahun, PELAB, Linköping University, Linköping, Sweden.

Anton Sodja, University of Ljubljana, Ljubljana, Slovenia

Ingo Staack, IEI, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.

Hubert Thierot, CEP Paristech, Ecole des Mines, Paris, France.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Parham Vasaiely, EADS, Hamburg, Germany.

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.

Robert Wotzlaw, Goettingen, Germany.

Azam Zia, PELAB, Linköping University, Linköping, Sweden.

30.5 OpenModelica Contributors 2011

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

Jens Frenkel, TU Dresden, Dresden, Germany.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Mikael Axin, IEI, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Robert Braun, IEI, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Stefan Brus, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy.

Filippo Donida, Politecnico di Milano, Milan, Italy.

Anand Ganeson, PELAB, Linköping University, Linköping, Sweden.

Mahder Gebremedhin, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Zoheb Hossain, PELAB, Linköping University, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden.

Kim Jansson, PELAB, Linköping University, Linköping, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.

Tommi Karhela, VTT, Espoo, Finland.

Joel Klinghed, PELAB, Linköping University, Linköping, Sweden.

Petter Krus, IEI, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland.

Abhinn Kothari, PELAB, Linköping University, Linköping, Sweden.

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.

Oliver Lenord, Siemens PLM, California, USA.

Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.

Rickard Lindberg, PELAB, Linköping University, Linköping, Sweden

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Henrik Magnusson, Linköping, Sweden.

Abhi Raj Metkar, CDAC, Trivandrum, Kerala, India.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Tuomas Miettinen, VTT, Espoo, Finland.

Afshin Moghadam, PELAB, Linköping University, Linköping, Sweden.

Maroun Nemer, CEP Paristech, Ecole des Mines, Paris, France.

Hannu Niemistö, VTT, Espoo, Finland.

Peter Nordin, IEI, Linköping University, Linköping, Sweden.

Kristoffer Norling, PELAB, Linköping University, Linköping, Sweden.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Karl Pettersson, IEI, Linköping University, Linköping, Sweden.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Reino Ruusu, VTT, Espoo, Finland.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Wladimir Schamai, EADS, Hamburg, Germany.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany.

Klas Sjöholm, PELAB, Linköping University, Linköping, Sweden.

Anton Sodja, University of Ljubljana, Ljubljana, Slovenia

Ingo Staack, IEI, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.

Hubert Thierot, CEP Paristech, Ecole des Mines, Paris, France.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Parham Vasaiely, EADS, Hamburg, Germany.

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.

Robert Wotzlaw, Goettingen, Germany.

Björn Zachrisson, MathCore Engineering AB, Linköping, Sweden.

Azam Zia, PELAB, Linköping University, Linköping, Sweden.

30.6 OpenModelica Contributors 2010

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Per Östlund, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Adeel Asghar, PELAB, Linköping University, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Simon Björklén, PELAB, Linköping University, Linköping, Sweden.

Mikael Blom, PELAB, Linköping University, Linköping, Sweden.

Robert Braun, IEI, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Stefan Brus, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy.

Filippo Donida, Politecnico di Milano, Milan, Italy.

Henrik Eriksson, PELAB, Linköping University, Linköping, Sweden.

Anders Fernström, PELAB, Linköping University, Linköping, Sweden.

Jens Frenkel, TU Dresden, Dresden, Germany.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden.

Kim Jansson, PELAB, Linköping University, Linköping, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany.

Tommi Karhela, VTT, Espoo, Finland.

Joel Klinghed, PELAB, Linköping University, Linköping, Sweden.

Petter Krus, IEI, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland.

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden.

Magnus Leksell, Linköping, Sweden.

Oliver Lenord, Bosch-Rexroth, Lohr am Main, Germany.

Ariel Liebman, Energy Users Association of Australia, Victoria, Australia.

Rickard Lindberg, PELAB, Linköping University, Linköping, Sweden

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Henrik Magnusson, Linköping, Sweden.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Hannu Niemistö, VTT, Espoo, Finland.

Peter Nordin, IEI, Linköping University, Linköping, Sweden.

Kristoffer Norling, PELAB, Linköping University, Linköping, Sweden.

Lennart Ochel, Fachhochschule Bielefeld, Bielefeld, Germany.

Atanas Pavlov, Munich, Germany.

Karl Pettersson, IEI, Linköping University, Linköping, Sweden.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Reino Ruusu, VTT, Espoo, Finland.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Wladimir Schamai, EADS, Hamburg, Germany.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany.

Klas Sjöholm, PELAB, Linköping University, Linköping, Sweden.

Anton Sodja, University of Ljubljana, Ljubljana, Slovenia

Ingo Staack, IEI, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Sonia Tariq, PELAB, Linköping University, Linköping, Sweden.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany.

Robert Wotzlaw, Goettingen, Germany.

Björn Zachrisson, MathCore Engineering AB, Linköping, Sweden.

30.7 OpenModelica Contributors 2009

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Constantin Belyaev, Bashpromavtomatika Ltd., Ufa, Russia

Simon Björklén, PELAB, Linköping University, Linköping, Sweden.

Mikael Blom, PELAB, Linköping University, Linköping, Sweden.

Willi Braun, Fachhochschule Bielefeld, Bielefeld, Germany.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Stefan Brus, PELAB, Linköping University, Linköping, Sweden.

Francesco Casella, Politecnico di Milano, Milan, Italy

Filippo Donida, Politecnico di Milano, Milan, Italy

Henrik Eriksson, PELAB, Linköping University, Linköping, Sweden.

Anders Fernström, PELAB, Linköping University, Linköping, Sweden.

Jens Frenkel, TU Dresden, Dresden, Germany.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Michael Hanke, NADA, KTH, Stockholm

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Alf Isaksson, ABB Corporate Research, Västerås, Sweden

Kim Jansson, PELAB, Linköping University, Linköping, Sweden.

Daniel Kanth, Bosch-Rexroth, Lohr am Main, Germany

Tommi Karhela, VTT, Espoo, Finland.

Joel Klinghed, PELAB, Linköping University, Linköping, Sweden.

Juha Kortelainen, VTT, Espoo, Finland

Alexey Lebedev, Equa Simulation AB, Stockholm, Sweden

Magnus Leksell, Linköping, Sweden

Oliver Lenord, Bosch-Rexroth, Lohr am Main, Germany

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Henrik Magnusson, Linköping, Sweden

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Hannu Niemistö, VTT, Espoo, Finland

Kristoffer Norling, PELAB, Linköping University, Linköping, Sweden.

Atanas Pavlov, Munich, Germany.

Pavol Privitzer, Institute of Pathological Physiology, Praha, Czech Republic.

Per Sahlin, Equa Simulation AB, Stockholm, Sweden.

Gerhard Schmitz, University of Hamburg, Hamburg, Germany

Klas Sjöholm, PELAB, Linköping University, Linköping, Sweden.

Martin Sjölund, PELAB, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Mohsen Torabzadeh-Tari, PELAB, Linköping University, Linköping, Sweden.

Niklas Worschech, Bosch-Rexroth, Lohr am Main, Germany

Robert Wotzlaw, Goettingen, Germany

Björn Zachrisson, MathCore Engineering AB, Linköping, Sweden

30.8 OpenModelica Contributors 2008

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Vasile Baluta, PELAB, Linköping University, Linköping, Sweden.

Mikael Blom, PELAB, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Henrik Eriksson, PELAB, Linköping University, Linköping, Sweden.

Anders Fernström, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Kim Jansson, PELAB, Linköping University, Linköping, Sweden.

Joel Klinghed, PELAB, Linköping University, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Kristoffer Norling, PELAB, Linköping University, Linköping, Sweden.

Anders Sandholm, PELAB, Linköping University, Linköping, Sweden.

Klas Sjöholm, PELAB, Linköping University, Linköping, Sweden.

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Simon Bjorklén, PELAB, Linköping University, Linköping, Sweden.

Constantin Belyaev, Bashpromavtomatika Ltd., Ufa, Russia

30.9 OpenModelica Contributors 2007

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Henrik Eriksson, PELAB, Linköping University, Linköping, Sweden.

Anders Fernström, PELAB, Linköping University, Linköping, Sweden.

Pavel Grozman, Equa AB, Stockholm, Sweden.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Ola Leifler, IDA, Linköping University, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Eric Meyers, Pratt & Whitney Rocketdyne, Palm City, Florida, USA.

Kristoffer Norling, PELAB, Linköping University, Linköping, Sweden.

Anders Sandholm, PELAB, Linköping University, Linköping, Sweden.

Klas Sjöholm, PELAB, Linköping University, Linköping, Sweden.

William Spinelli, Politecnico di Milano, Milano, Italy

Kristian Stavåker, PELAB, Linköping University, Linköping, Sweden.

Stefan Vorkoetter, MapleSoft, Waterloo, Canada.

Björn Zachrisson, MathCore Engineering AB, Linköping, Sweden.

Constantin Belyaev, Bashpromavtomatika Ltd., Ufa, Russia

30.10 OpenModelica Contributors 2006

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, MathCore Engineering AB, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

David Akhvlediani, PELAB, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Anders Fernström, PELAB, Linköping University, Linköping, Sweden.

Elmir Jagudin, PELAB, Linköping University, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Kaj Nyström, PELAB, Linköping University, Linköping, Sweden.

Lucian Popescu, MathCore Engineering AB, Linköping, Sweden.

Andreas Remar, PELAB, Linköping University, Linköping, Sweden.

Anders Sandholm, PELAB, Linköping University, Linköping, Sweden.

30.11 OpenModelica Contributors 2005

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, PELAB, Linköping University and MathCore Engineering AB, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Ingemar Axelsson, PELAB, Linköping University, Linköping, Sweden.

David Broman, PELAB, Linköping University, Linköping, Sweden.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Kaj Nyström, PELAB, Linköping University, Linköping, Sweden.

Lucian Popescu, MathCore Engineering AB, Linköping, Sweden.

Levon Saldamli, PELAB, Linköping University, Linköping, Sweden.

30.12 OpenModelica Contributors 2004

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, Linköping University, Linköping, Sweden.

Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.

Peter Bunus, PELAB, Linköping University, Linköping, Sweden.

Daniel Hedberg, MathCore Engineering AB, Linköping, Sweden.

Håkan Lundvall, PELAB, Linköping University, Linköping, Sweden.

Emma Larsdotter Nilsson, PELAB, Linköping University, Linköping, Sweden.

Kaj Nyström, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Lucian Popescu, MathCore Engineering AB, Linköping, Sweden.

Levon Saldamli, PELAB, Linköping University, Linköping, Sweden.

30.13 OpenModelica Contributors 2003

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, Linköping University, Linköping, Sweden.

Levon Saldamli, PELAB, Linköping University, Linköping, Sweden.

Peter Bunus, PELAB, Linköping University, Linköping, Sweden.

Vadim Engelson, PELAB, Linköping University, Linköping, Sweden.

Daniel Hedberg, Linköping University, Linköping, Sweden.

Eva-Lena Lengquist-Sandelin, PELAB, Linköping University, Linköping, Sweden.

Susanna Monemar, PELAB, Linköping University, Linköping, Sweden.

Adrian Pop, PELAB, Linköping University, Linköping, Sweden.

Erik Svensson, MathCore Engineering AB, Linköping, Sweden.

30.14 OpenModelica Contributors 2002

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Levon Saldamli, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, Linköping University, Linköping, Sweden.

Daniel Hedberg, Linköping University, Linköping, Sweden.

Henrik Johansson, PELAB, Linköping University, Linköping, Sweden

Andreas Karström, PELAB, Linköping University, Linköping, Sweden

30.15 OpenModelica Contributors 2001

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

Levon Saldamli, PELAB, Linköping University, Linköping, Sweden.

Peter Aronsson, Linköping University, Linköping, Sweden.

30.16 OpenModelica Contributors 2000

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

30.17 OpenModelica Contributors 1999

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden

Peter Rönnquist, PELAB, Linköping University, Linköping, Sweden.

30.18 OpenModelica Contributors 1998

Peter Fritzson, PELAB, Linköping University, Linköping, Sweden.

David Kågedal, PELAB, Linköping University, Linköping, Sweden.

Vadim Engelson, PELAB, Linköping University, Linköping, Sweden.

関連図書

[CK06] Francois E. Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387261028.

[DN10] T. A. Davis and E. Palamadai Natarajan. Algorithm 907: klu, a direct sparse solver for circuit simulation problems. *ACM Trans. Math. Softw.*, 37(3):36:1–36:17, September 2010. URL: http://doi.acm.org/10.1145/1824801.1824814, doi:10.1145/1824801.1824814.

[HNorsettW93] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Non-stiff Problems*. Springer-Verlag Berlin Heidelberg, 2nd rev. ed. 1993. corr. 3rd printing 2008 edition, 1993. ISBN 978-3-540-56670-0. doi:10.1007/978-3-540-78862-1.

[HBG+05] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. SUNDIALS: suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

[Pet82] L.R. Petzold. Description of dassl: a differential/algebraic system solver. 1982.

[Axe05] Ingemar Axelsson. OpenModelica Notebook for interactive structured Modelica documents. Master's thesis, Linköping University, Department of Computer and Information Science, October 2005. LITH-IDA-EX–05/080–SE.

[Fernstrom06] Anders Fernström. Extending OpenModelica Notebook – an interactive notebook for structured Modelica documents. Master's thesis, Linköping University, Department of Computer and Information Science, September 2006. LITH-IDA-EX–06/057—SE.

[Fri04] Peter Fritzson. *Principles of Ob - ject-Ori - ent - ed Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, February 2004. ISBN 0-471-471631.

[Knu84] Donald E. Knuth. Literate programming. *The Computer Journal*, 27:97–111, 1984.

[Wol96] Stephen Wolfram. *The Mathematica Book*. Wolfram Media/Cambridge University Press, third edition, 1996.

[BOR+12] Bernhard Bachmann, Lennart Ochel, Vitalij Ruge, Mahder Gebremedhin, Peter Fritzson, Vaheed Nezhadali, Lars Eriksson, and Martin Sivertsson. Parallel multiple-shooting and collocation Optimization with OpenModelica. In Martin Otter and Dirk Zimmer, editors, *Proceedings of the 9th International Modelica Conference*. Linköping University Electronic Press, September 2012. doi:10.3384/ecp12076659.

[RBB+14] Vitalij Ruge, Willi Braun, Bernhard Bachmann, Andrea Walther, and Kshitij Kulshreshtha. Efficient implementation of collocation methods for optimization using openmodelica and adol-c. In Hubertus Tummescheit and Karl-Erik Årzén, editors, *Proceedings of the 10th International Modelica Conference*. Modelica Association and Linköping University Electronic Press, March 2014. doi:10.3384/ecp140961017.